Published by the IEEE Computer Society
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1264

IEEE Computer Society Press Order Number P2772
Library of Congress Number 2006939402
ISBN 0-7695-2772-8

ISBN 0-7695-2772-8
90000
9 780769 527727

14th IEEE International Conference and Workshops on the **Engineering of Computer-Based Systems**

IEEE COMPUTER SOCIETY

IEEE

Proceedings

14th IEEE International Conference and Workshops on the

Engineering of Computer-Based Systems

ECBS

Raising Expectations of Computer-Based Systems

IEEE COMPUTER SOCIETY

IEEE

Tucson, Arizona
26–29 March 2007

Sponsored by
IEEE Technical Committee on Engineering of Computer-Based Systems

Proceedings

14th Annual IEEE International Conference and Workshops on the

Engineering of Computer-Based Systems

ECBS 2007

Raising Expectations of Computer-Based Systems

IEEE

Proceedings

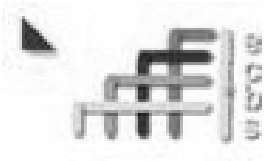
14th Annual IEEE International Conference and Workshops on the

Engineering of Computer-Based Systems

ECBS 2007

Raising Expectations of Computer-Based Systems

March 26-29, 2007
Tucson, Arizona



Edited by
John Leaney
Jerzy W. Rozenblit
Jianfeng Peng

Sponsored by
IEEE Computer Society

Hosted by
The University of Arizona

Los Alamitos, California

Washington • Tokyo

IEEE Computer Society Order Number P2772
ISBN 0-7695-2772-8
ISBN 978-0-7695-2772-7
Library of Congress Number 2006939402

Additional copies may be ordered from:

IEEE Computer Society
Customer Service Center
10662 Los Vaqueros Circle
P.O. Box 3014
Los Alamitos, CA 90720-1314
Tel: + 1 800 272 6657
Fax: + 1 714 821 4641
http://computer.org/cspress
csbooks@computer.org

IEEE Service Center
445 Hoes Lane
P.O. Box 1331
Piscataway, NJ 08855-1331
Tel: + 1 732 981 0060
Fax: + 1 732 981 9667
http://shop.ieee.org/store/
customer-service@ieee.org

IEEE Computer Society
Asia/Pacific Office
Watanabe Bldg., 1-4-2
Minami-Aoyama
Minato-ku, Tokyo 107-0062
JAPAN
Tel: + 81 3 3408 3118
Fax: + 81 3 3408 3553
tokyo.ofc@computer.org

Individual paper REPRINTS may be ordered at: <reprints@computer.org>

Editorial production by Lisa O'Conner
Cover art production by Joe Daigle/Studio Productions
Printed in the United States of America by Applied Digital Imaging

Conference Publishing Services
http://www.computer.org/proceedings/

Table of Contents

14th Annual IEEE International Conference and Workshop
on the Engineering of Computer-Based Systems

ECBS 2007

Keynote Address

Session A1: Architectures

Session A2: Architectures

Session A3: Architectures

Session B1: Component-Based System Design

Session B2: Distributed Systems Design

Session B3: Distributed Systems Design

Session C1: Embedded Real-Time Software Systems

Session C2: Embedded Real-Time Software Systems

Session C3: Embedded Real-Time Software Systems

Session A4: Lifecycle Processes and Process Evolution

Session A5: Requirements Elicitation and Analysis

Session A6: Model-Based System Development

Session B4: Medical Applications

Session B5: Modeling and Analysis of Complex Systems

Session B6: Reengineering and Reuse

Session C4: Industrial Applications

Session C5: Reliability, Safety, Dependability, Security

Session C6: Reliability, Safety, Dependability, Security

Session A7: Doctoral Symposium

Session B7: System Assessment, Testing, and Metrics

Session B8: System Assessment, Testing, and Metrics

Session C7: Verification and Validation

Session C8: Verification and Validation

ECBS Model-Based System Development Workshop

Poster

Foreword

The theme of this year's ECBS, "Raising Expectations of Computer Based Systems" was chosen partly in response to yet another summary of the overall poor state of software systems in terms of budget, delivery and performance.[1]

Computer based systems, despite their increased complexity, fair no worse than 'pure' software systems. That is the good news. The bad news is obvious; we are not getting better.

How can we raise expectations? Can we expect systems to operate reliably, to perform in a timely manner? Can we expect systems to manage functional change? We need to expect systems to be delivered, in a useable state, on time and within budget. As purchasers or developers, we may work together to improve this situation.

The accepted papers show a definite commitment to raising expectations. We hope that the forum of industrialists, government, and academics which is the ECBS community will come out of this conference having had meaningful discussions, and with a strong sense of direction to raise expectations of computer based systems.

We want to thank the authors for their interest and contributions to the success of this meeting. Our sincere gratitude goes to the referees who, as often is the case, were asked to review many papers in a relatively short time. Their rigor in assessing the quality is reflected in this volume. IEEE Computer Society Press deserves much credit for its professionalism in handing the printing process. Last but not least, we are delighted to welcome you back to sunny Tucson where ECBS was held in 1995. We look forward to an exciting, vibrant, and productive conference.

John Leaney, Jerzy W. Rozenblit, and Jianfeng Peng

---

[1] "The UK public sector alone has spent an estimated £12.4 billion on software in the last year and the overall UK spend on IT is projected to be a monumental £22.6 billion," says Basil Butler, Chairman of the working group that produced the report. "We looked at a range of studies showing that only around 16 per cent of IT projects can be considered truly successful." The challenge of Complex IT projects (in the UK), http://www.bcs.org/server.php?show=conWebDoc.1761

Robert L. Glass, "Looking into the challenges of complex IT projects", *Communications of the ACM*, Volume 49 Issue 11, November 2006.

Conference Organization

General Chair
Jerzy W. Rozenblit

Program Co-Chairs
John Leaney, Tim O'Neill, Jianfeng Peng

Asia Pacific Chair
Tim O'Neill, John Leaney

Europe Co-Chairs
Matthias Riebisch, Peter Tabeling

Publicity Chair
Brian Ten Eyck

Local Arrangements Chair
Rozanne Canizales

Steering Committee

Jerzy Rozenblit, Jianfeng Peng, John Leaney, Matthias Riebisch, Peter Tabeling , Stephanie White, Byron Purves, Roy Sterritt, Jonah Lavi, Miroslav Sveda, Wilhelm Rossak

Conference Organization

General Chair
Jerzy W. Rozenblit

Program Co-Chairs
John Leaney, Tim O'Neill, Jianfeng Peng

Asia Pacific Chair
Tim O'Neill, John Leaney

Europe Co-Chairs
Matthias Riebisch, Peter Tabeling

Publicity Chair
Brian Ten Eyck

Local Arrangements Chair
Rozanne Canizales

Steering Committee

Jerzy Rozenblit, Jianfeng Peng, John Leaney, Matthias Riebisch, Peter Tabeling , Stephanie White, Byron Purves, Roy Sterritt, Jonah Lavi, Miroslav Sveda, Wilhelm Rossak

Program Committee

Shane D. Arnott	*Australia*
Ted Bapty	*USA*
Fernando Barros	*Portugal*
Agostino Bruzzone	*Italy*
Klaus Buchenrieder	*Germany*
Piotr Czapiewski	*USA*
Armin Eberlein	*Canada*
Bernhard Gröne	*Germany*
Toni Guasch	*Spain*
Witold Jacak	*Austria*
Gabor Karsai	*USA*
Ryszard Klempous	*Poland*
Jonah Lavi	*Israel*
Bud Lawson	*Sweden*
John Leaney	*Australia*
Andrew Lyons	*Germany*
Roman Lysecky	*USA*
Susan Lysecky	*USA*
Mike Mannion	*UK*
Bonnie Melhart	*USA*
Elena Navarro Martinez	*Spain*
Jan Nikodem	*Poland*
Tim O'Neill	*Australia*
Dietmar Pfahl	*Germany*
Miquel Àngel Piera Eroles	*Spain*
Miroslav Popovic	*Serbia*
Byron Purves	*USA*
Matthias Riebisch	*Germany*
Wilhelm Rossak	*Germany*
Jerzy Rozenblit	*USA*
Stefano Saetta	*Italy*
Bernhard Schätz	*Germany*
Stephan Schulz	*France*
Alexander Sedlmaier	*Germany*
Roy Sterritt	*Northern Ireland*
Miroslav Sveda	*Czech Republic*
Peter Tabeling	*Germany*
Claudio Talarico	*USA*
Stephanie White	*USA*
Heinz Züllighoven	*Germany*

Reviewers

Shane Arnott	*Australia*
Ted Bapty	*USA*
Fernando Barros	*Portugal*
Peter Braun	*Germany*
Klaus Buchenrieder	*Germany*
David Bustard	*UK*
Piotr Czapiewski	*USA*
Mark Denford	*Australia*
Armin Eberlein	*United Arab Emirates*
Christian Erfurth	*Germany*
Tony Ewing	*USA*
Chuan Feng	*USA*
Reuven Gallant	*Israel*
Vahid Garousi	*Canada*
Holger Giese	*Germany*
Bernhard Gröne	*Germany*
Hanees Haniffa	*USA*
Gabor Karsai	*USA*
Ryszard Klempous	*Poland*
Heike Kraef	*Germany*
Jonah Lavi	*Israel*
Harold Lawson	*Sweden*
John Leaney	*Australia*
Andreas Liehr	*Germany*
Andrew Lyons	*Germany*
Susan Lysecky	*USA*
Roman Lysecky	*USA*
Mike Mannion	*Scotland*
Petr Matousek	*Czech Republic*
Bonnie Melhart	*USA*
Faisal Momen	*USA*
Elena Navarro	*Spain*
Jan Nikodem	*Poland*
Tim O'Neill	*Australia*
Dwight Peltzer	*USA*
Jianfeng Peng	*USA*
Dietmar Pfahl	*Canada*
Jan Philipps	*Germany*
Miquel Angel Piera Eroles	*Spain*
Miroslav Popovic	*Serbia*
Alexander Pretschner	*Switzerland*
Byron Purves	*USA*
Haiyan Qiao	*USA*
Juan José Ramos	*Spain*
Matthias Riebisch	*Germany*

Wilhelm Rossak	*Germany*
Jerzy Rozenblit	*USA*
Ondrej Rysavy	*Czech Republic*
Bernhard Schätz	*Germany*
Stephan Schulz	*France*
Roy Sterritt	*Northern Ireland*
Miroslav Sveda	*Czech Republic*
Claudio Talarico	*USA*
Stephanie White	*USA*
Sven Wohlfarth	*Germany*
Heinz Zuellighoven	*Germany*

Composition of Cyber-Physical Systems

Janos Sztipanovits
Institute for Software Integrated Systems, Vanderbilt University
Nashville, TN, USA
janos.sztipanovits@vanderbilt.edu

Abstract

Cyber Physical Systems (CPS) is emerging as a new research discipline at the intersection of physical, biological, engineering and information sciences. Information technology, once a provider of tools for sciences and engineering, has become an interdisciplinary field at the center of scientific and industrial innovation. This talk addresses some of the basic challenges of CPS and its relationship to ECBS.

INTRODUCTION

During the first 40 years of computing, physical and information sciences evolved along different trajectories. Dominating abstractions in Computer Science avoided physical characteristics of information processing, while systems science emerged as a mathematical foundation for modeling of physical processes. However, new trends of the past fifteen years have made this separation strongly limiting. In life sciences, the emerging discipline of bioinformatics has shown the significance of information science abstractions in understanding biological processes. In engineering, computing and communication have become 'universal system integrator' for physical systems making computer based engineering systems the major source of industrial innovation. Complex engineering systems are now designed and built by composing physical and computational objects. These developments have pointed toward the need of reintegrating physical and information sciences that requires addressing the challenges of developing a new scientific foundation for CPS.

DESIGN CHALLENGES FOR CPS

Design and implementation of complex systems increasingly require the involvement of three major disciplines: control, systems and software engineering. This is not necessarily an indication of increasing design complexity, since decomposing a design problem into orthogonal aspects is our primary tool for managing and decreasing complexity. Orthogonal abstractions allow the separation of design concerns, thereby decoupling complicated, intertwined design flows. However, it is also recognized that these three dominant aspects of modern system design do not offer orthogonal abstractions for CPS-s, therefore do not provide good foundation for separation of design concerns. They rather reflect traditional disciplinary and organizational boundaries. Decisions in control, software and systems engineering aspects of CPS-s have strong interdependences and this limits compositionality in the design process.

Developing, modifying and integrating abstractions that cover CPS design spaces represent one of the major challenges. In control and systems engineering, abstractions are traditionally expressed using models. The increased use of design automation tools has led to the introduction of a wide range of modeling languages – usually linked to specific analysis and simulation tools, (such as Simulink/Stateflow or Modellica) or to various domain specific standards (such as DODAF or SySML).

During the last decade, driven by the increasing popularity of object-oriented programming languages, modeling has become the mainstream in software engineering as well. Research directions, such as Model Integrated Computing (MIC), Model Driven Architecture (MDA), Model Driven Design (MDD), Business Process Modeling (BPM) and others made model-based approaches in software engineering practical and led to the appearance of a new generation of CASE tools. However, the most important consequence of these developments have been the opportunity for convergence in control, systems and software engineering built on the principles of model-based design.

Model-based design uses models, which are formal, composable and manipulable during the design process. The key breakthrough in the early success of model-based design has been the emergence of metamodeling and metaprogrammable tool suites that enabled the affordable introduction of modeling languages that are domain-specific, offering systems designers modeling concepts and notations that are tailored to characteristics of their application domain.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

The need for domain specificity is a natural consequence of the inherent heterogeneity of CPS. The behavior of physical and computational components need to be represented using abstractions supporting required analysis and synthesis tasks. These abstractions are determined by the nature of the underlying physics (or biology), the required level of fidelity in modeling dynamic behaviors and the scope of essential physical behaviors, such as power consumption and size. Abstractions need to be developed for modeling interactions among components. Naturally, there is essential difference between abstractions used for modeling physical interactions and abstractions used for modeling component interactions based on digital communication mechanisms.

This talk will review fundamental problems and promising approaches for composing heterogeneous CPS.

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Generation of Related Performance Simulation Models at an Early Stage in the Design Cycle

Andreas W. Liehr, Klaus Buchenrieder
Institut für Technische Informatik
Universität der Bundeswehr München
D-85577 Neubiberg, Germany
{*andreas.liehr*|*klaus.buchenrieder*}*@unibw.de*

Abstract

In the early development stages of embedded systems, fundamental decisions concerning the system architecture and a proper hardware / software partition must be made. The choices determine, e.g. power consumption, performance, and other system features. For this reason, a method plus supporting tools for the fast, efficient, and automated evaluation of system design alternatives is required.

In this contribution, we present the MIRA Framework and a novel method based on Extended Queuing Network Models (EQN) linked to a set of simulation models for architectural exploration and hardware / software tradeoff analysis. The MIRA Framework consits of hardware components expressed in XML and function represented as UML Activity Diagrams. Activity threads, introduced in this contribution, define building rules for the simulation models. This method fosters the combined simulation and analysis of system behavior as well as architectural exploration of design alternatives.

As current experimental work and evaluation with nontrivial examples show, the presented method allows for fast and efficient generation of simulation models and an accurate prediction of performance measures.

1. Introduction

The discovery of the product form for Queuing Network Models (QNM), their properties and the development of efficient computational algorithms to determine the product form by Baskett et al. [4], is regarded as a breakthrough in analytical performance modeling [16].

Numerous improvements [2] and approaches to extend QNMs have been made [25, 26]. Sauer introduced passive queues to QNM. While a job of a QNM can only be member of one active node, a job of an EQN can be member of one active and an arbitrary number of passive queues. The advantage of this approach, compared to ordinary QNM, is the possibility for a job to use more than one system resource at a time. This provides the ability to generate more realistic system models than with the original QNM.

The usability of QNMs for performance evaluation is limited by the fact, that developers are required to have profound knowledge of performance modeling. Besides that, it would mean a significant effort to build QNMs for complex systems manually. Those aspects motivated several works concerning the supported generation of QNM in the last years.

The Software Performance Engineering Method, first introduced by Smith [27], consists of two models. The software execution model takes the form of execution graphs and represents the *software* behavior. The second model, the system execution model, is based on QNM. It describes the *system* behavior. The input data of the second model is a combination of the results from the software execution model and information about the system hardware. Extensions of this approach have been published by Cortellessa et al. [7, 8].

Another group of approaches is based upon pattern based methods. In those, the developer defines system models from predefined and reusable system patterns. After that, the simulation model will be generated from the pattern based system. It is a very efficient and fast system, but a pattern definition is required for every architectural class. Approaches of this method have been published by Gomaa and Menasé [11], Petriu et al. [23, 24], Balsamo et al. [3].

A third group of approaches are entirely Unified Modeling Language (UML) based development tools. The advantage of those models is the high popularity of UML. Therefore, UML is well known by most system developers [35]. In these approaches, a designer specifies a computer system in UML and annotates the performance values. The tool generates QNM directly from the UML model without any

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

user interaction. Such tools were introduced by Kabajunga and Pooley [18], Xu and Lehmann [35, 36, 37], Käkipuro [19], Gu and Petriu [13, 14, 15], and Woodside [34].

In all those works the focus lies on the modeling and solving of a single simulation models representing one design approach. For the evaluation of related design approaches, each single approach has to be simulated.

System development tools, that consider the requirement of the evaluation of related approaches, exist for example from Cadence. The Cadence - Virtual Component Codesign tool VCC is described in [6, 10]. Such tools aim at a more detailed simulation model at a later state of the system development process. Thus, they do not enable fast experimentation concerning the feasibility of different design approaches.

In this work, we present a method that uses a UML based software representation and a pattern based hardware model. Additionally, we introduce activity threads, that map functional components to the hardware architecture. Those activity threads enable us to build several EQN for alternative system configurations from one software model and one hardware model. This provides a fast and efficient method for the performance evaluation of different approaches for the same computer system. Differences may concern, for example the partitioning into hardware and software, different scheduling algorithms or different types of CPUs.

This paper is organized as follows: Section 2 introduces the proposed design method. In Section 3 we present the MIRA Framework that uses the stated approach for performance evaluation. Section 4 shows experimental results with the MIRA Framework. The paper concludes with summary and outlook.

2. Generating Simulation Models for Related Design Approaches

The proposed approach relies on three models to generate the complete simulation model.

The hardware model defines the system hardware architecture for the planned computer system. It has to be a superset of the required hardware for all system design approaches to be evaluated. Depending on the considered design approach, only a subset of the whole hardware model has to be taken into account. To fulfill our objective to build a fast and easy to use system modeling tool, we chose a pattern based design approach for our hardware model. For this purpose, a hardware pattern database with components, classified as bus, bus bridge, CPU, logic block and memory, has been utilized. Each pattern contains specific performance information and interface definitions. The user chooses the required components from the database and builds a hardware model out of them. This model will be enriched with the data from the component database and stored as an XML File. The hardware components CPU and logic block are grouped into worker classes. These define CPUs or logic blocks, for which the same amount of instructions or the same delay approach concerning one functional definition applies. This comes to relevance by the definition of the software model in the next paragraph.

The software model contains the performance related information concerning software components of the computer system to be designed. The term software component is used in the meaning of system component as stated in [12]. The model is split into two components. The first one represents the control flow of the intended functionality of our system as UML activity diagram [22]. The second defines performance relevant information for every activity of the activity diagram. Such information comprises the workload of a CPU, the delay for the execution on a logic block, and the IO behavior of the software component. Depending on the intended alternatives to evaluate, for each activity exist *1-n* blocks of performance relevant information. Thereby, n symbols the number of different worker class, defined in the hardware model. This software model has to be a superset of the software components used in all design approaches and each single approach has to use a subset of the software components from this software model.

The third model links the functional descriptions from the software model to hardware components. To accomplish that, we introduce activity threads. Activity threads are realized as graphs with a start node and an end note. Each path from the start node to the end note contains m intermediate nodes. Thereby, m represents the number of activities in the activity diagram of the software model. Such a path represents one design method to evaluate. The number of differing paths is the number of system models to build.

After the user has supplied the three models, the system model generation works without user interaction. The QNMs are built from predefined patterns. A pattern must exist for every kind of worker class, memory, bus and bus bridge that is available in the hardware pattern database. The functional behavior is modeled with global variables, job variables and set nodes. Those components influence the routing behavior of jobs, the delay of active nodes and the number of tokens to acquire from passive nodes.

The result of this method is a set of EQNs, representing the related design approaches. The simulation of them can be realized with off the shelf tools. The usage of EQN for the simulation models enables the simulation of the simultaneous posession of multiple system resources. That gives us the opportunity to predict the bahavior of the system, when several processes compete for ressources like busses or memory.

3. The MIRA Framework

To build those QNMs, stated above, we introduce the Model based Investigation of Related Architectures (MIRA) Framework. At this stage of our work, the user must specify the models according to the defined syntax for the XML representation by hand. In future revisions, a tool supported model creation will become available.

3.1. Hardware Model

As stated in the previous section, hardware is represented by a pattern based model, which consists of predefined hardware components. Such a component may represent a bus, a bus bridge, a CPU, a logic block or a memory. With those classes of hardware components, arbitrary classes of computer system can be represented [6]. Those predefined patterns consist of interface definitions and annotations concerning performance. Hardware patterns are stored in a hardware pattern database, so that patterns can be added more easily in the future. Generally, the database contains all hardware based performance information, required to build a system model. The information can be obtained from the manufacturer or by experiments. For our measurement, the hardware pattern database contained only few hardware components. The performance information reflected benchmarking results from real-system tests.

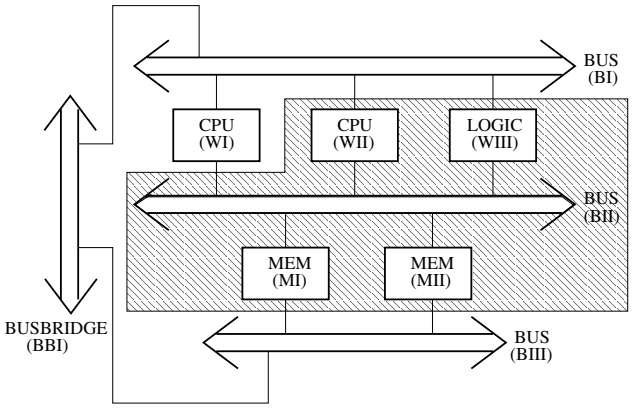


Figure 1. Hardware Example 1

Depending on the implementation of the system to evaluate, only a part of this hardware will be used for the system model. As an example, consider Figure 1, that depicts Hardware Example 1 (HWE1), an architecture consisting of a CPU (WII), logic (WIII), two memory elements (MI,MII) and one bus (BII) as contained in the highlighted area. The definition of the hardware model is represented by an XML file. Listing 1 shows the representation of the hardware model visualized above. We also defined a DTD Definition [30] for the XML hardware representation model, so it is possible to proof its syntactical correctness by a validator like Validome [29].

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hwm SYSTEM "http://localhost/xml/mira.dtd">
<hardwaremodel>
  <cpu id="WI" class="I" hwid="1001"/>
  <cpu id="WII" class="I" hwid="1002"/>
  <logic id="WIII" class="II" hwid="2001"/>
  <memory id="MI" hwid="3001"/>
  <memory id="MII" hwid="3002"/>
  <bus id="BI" hwid="4001" protocol="FCFS">
    <connection component="WI"/>
    <connection component="WII"/>
    <connection component="WIII"/>
  </bus>
  <bus id="BII" hwid="4001" protocol="FCFS">
    <connection component="WI"/>
    <connection component="WII"/>
    <connection component="WIII"/>
    <connection component="MI"/>
    <connection component="MII"/>
  </bus>
  <bus id="BIII" hwid="4002" protocol="FCFS">
    <connection component="MI"/>
    <connection component="MII"/>
  </bus>
  <busbridge id="BBI" hwid="5001">
    <connection component="BI"/>
    <connection component="BIII"/>
  </busbridge>
</hardwaremodel>
```

Listing 1. XML Representation of Hardware Example 1

Each component is represented by an XML element, that contains an unique id attribute named `id` and an attribute `hwid`. The unique id is used to define the bus connections. For that purpose, a nested element named connection inside the bus element is used. It contains an attribute `component` that references the unique id of a CPU, a logic block or a memory. Bus bridges are used to interconnect bus elements. The layout of the bus bridge element is similar to the bus element, but the nested connection elements are only allowed to reference to busses.

The XML representation of the hardware model contains no performance information. Each element contains the attribute `hwid`, that references an entry of the hardware pattern database, representing this hardware component. The performance specific data will be queried from the hardware pattern database in the building process of the EQNs for the performance simulation.

Currently, the hardware pattern database is realized as an XML file, containing one entry for each known hardware pattern. Listing 2 shows a short exemplaric extract from the hardware pattern database.

Scheduling policies for CPUs and busses are considered as part of the hardware and have to be defined in the hardware pattern database. Therefore, one CPU with two possible scheduling algorithms has two entries in the hardware

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

pattern database, so that it can be referenced by two different values for the attribute `hwid`.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hwm SYSTEM "http://localhost/xml/mira.dtd">
<hardwaredatabase>
  <cpu hwid="1002">
    <core type="1" speed="1000"/>
    <cache type="1" size="524288"/>
    ...
  </cpu>
  ...
</hardwaredatabase>
```

Listing 2. Example Entry from the Hardware Pattern Database

3.2. Software Model

Inside the MIRA Framework, software is represented by a functional definition and fitting performance information. The functional part utilizes activity diagrams to define the intended behavior of the system to be designed. An example is given in Figure 2.

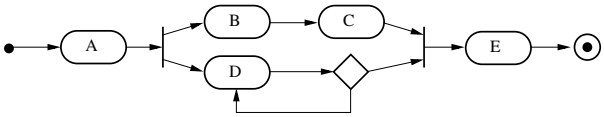


Figure 2. Example for Functional Behavior, rRepresented as UML Activity Chart

The activity diagram model supports forks, joins, conditions and decisions. Each activity will be labelled with a unique identifier. This model is compliant to the UML 2.0 specification. Thus, it is possible to design the activity diagram model with a standard UML modeling tool that supports export of XMI (XML). We chose the free XML modeling tool Umbrello [28] for our work. Unfortunately, every XML modeling tool uses its own format for its XML representations of activity diagram models. Such a representation contains proprietary information, like the position of Elements in the GUI representation or links to other UML models. Hence, the output of the tool has to be translated to the MIRA framework specific XML representation of the activity diagram. We utilize a Perl based converter for this purpose. Such a specific converter is required for every UML modeling tool intended to use. A converter fitting for Umbrello is part of the MIRA framework.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE hwm SYSTEM "http://localhost/xml/mira.dtd">
<activitychart>
  <component type="start" id="1"/>
  <component type="stop" id="2"/>
  <component type="bar" id="3"/>
  <component type="bar" id="4"/>
  <component type="desicion" id="5"/>
  <component type="activity" id="6" name="A"/>
  <component type="activity" id="7" name="B"/>
  <component type="activity" id="8" name="C"/>
  <component type="activity" id="9" name="D"/>
  <component type="activity" id="10" name="E"/>
  <connection from="1" to="6"/>
  <connection from="6" to="3"/>
  <connection from="3" to="7"/>
  <connection from="3" to="9"/>
  <connection from="7" to="8"/>
  <connection from="8" to="4"/>
  <connection from="9" to="5"/>
  <connection from="5" to="9" condition=""/>
  <connection from="5" to="4" condition=""/>
  <connection from="4" to="10"/>
  <connection from="10" to="2"/>
</activitychart>
```

Listing 3. XML Representation for the Software Example

The XML representation of the example software from Figure 2 is shown in Listing 3. It contains an element for each activity diagram model component. Those components can stem from the type `start`, `stop`, `bar`, `decision` or from an `activity`. The type is defined by an attribute, named `type`. Each component has also a unique identifier attribute called `id`. The component elements from the type `activity` also contain a unique identifier called `name`. This identifier is used as reference id on the performance information part of the software model. Additionally, the XML model contains a set of connection elements. Each of these elements represent one connection in the activity diagram and contain the attribute `from` that references the starting component of the connection and the attribute `to` that references the target component of the connection.

Table 1. Example for Software Performance Information

ID	Worker Class	Instructions/ Delay	I/O	Mem.
A	I	$2k + P\left(\frac{1}{2}\right) * 500$	5%	1MB
A	II	$5\mu s$	5%	1MB
B	I	$12k$	15%	120kB
C	II	$2\mu s$	50%	0

The performance information part of the system model is built as a list with two identifiers. The first is the fitting activity from the activity diagram model, the second is the type of CPU or logic this performance information is fitting to.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

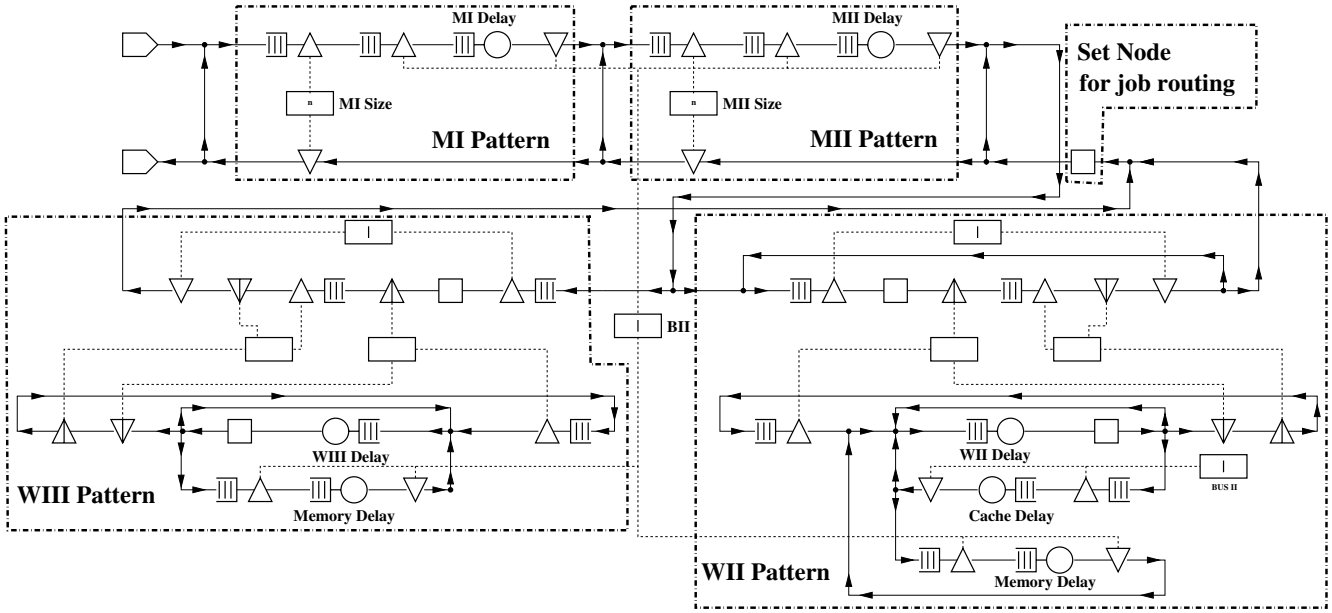


Figure 3. EQN Simulation Model for HWE1

The performance list in Table 1 shows an extract of a complete definition for software performance information. For each activity of the activity diagram model, at least one definition must exist. Depending on the intended simulation models, the list can have up to $(C * A)$ entries, where C represents the number of the worker class and A the number of activities defined in the activity diagram.

3.3. Activity Threads

The software model and the hardware model, are linked to system designs by activity threads. Those activity threads allow for the automated generation of numerous EQN based simulation models from one software model and one hardware model. Their number depends on the amount of alternatives, constructed in the befitting instance of the software model and the hardware model described above. For this purpose, at least one activity thread has to be defined.

Activity threads are represented as UML activity diagram model. Each path through the activity thread from the initial node to the final node represents a design approach for which an system model has to be built. Therefore, an activity defines the hardware components, which must be used to execute the intended functionality. So each activity is required to contain exactly one worker, at least one memory and a bus connection between the worker and all memory components used in this activity. Alternative bus connections inside one activity are not allowed. This prevents the distinct generation of the system model.

Figure 4 shows two exemplaric activity threads, for Hardware Example 1, introduced in Figure 1 and the software model from Figure 2. The upper activity thread contains one segment with two alternative routes. This results in two system models. The lower activity thread contains two segments, each with two alternative routes. From this activity thread will four system models be built..

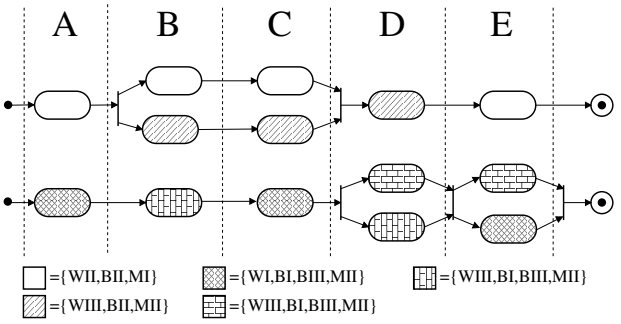


Figure 4. Activity Threads for the Example System

3.4. Building the Simulation Models

With these three models, the MIRA Framework is enabled to automatically build the system models. The following steps will be performed:

First, each path through the activity thread is converted to a new activity thread. To prevent irritations, we call those

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

new activity threads simple activity threads. One simple activity thread contains only one path from its start node to its end node. This step splits the two activity threads, shown in Figure 4 into six simple activity threads. For every simple activity thread, one simulation model is built. The following steps consider only one such simple activity thread, because the Framework handles all in the same way.

Second, the simple activity thread is parsed for hardware components. From this results a set of hardware components, which contains only components required by this system configuration. Together with the information from the hardware pattern database, the EQN representing this architecture is constructed from predefined patterns.

An example, consider the EQN shown in Figure 3. It represents HWE1. The CPU WII is realized with a Round Robin Scheduler and utilizes a Level-1 cache. The logic block $WIII$ employs a First Come First Served Scheduler. The delay models for the active queues, representing memory, bus, and CPU delay are generated from the component information in the hardware pattern database.

The resulting EQN matches the hardware architecture, to be simulated. At this stage, the model does not contain behavioral information concerning the functionality of the computer system.

This information is gathered from the simple activity thread and the second part of the software model. The IO behavior is mapped as routing decision inside the worker pattern. The memory utilization is represented by the number of tokens, the job acquires from the passive queue of the fitting memory pattern. The execution of the software activity on a logic block is represented by a deterministic delay. This delay is applied in the active queue called logic delay inside the worker pattern. The execution of the software on a particular CPU is modeled as functional delay with hardware parameters as variables. This delay is used in the active queue called CPU delay inside the fitting worker pattern.

The resulting EQN can now be converted from the XML representation, created by the MIRA Framework, to the syntax for an off-the-shelf simulation tool Arena [1] from Rockwell Automation.

4. Experiments

In our experiments we compare performance measures of four hardware configurations with the estimation results, obtained with the proposed MIRA framework. The reference hardware system consists of an 800MHz Pentium III CPU (Coppermine) equipped with a Level-1 data cache and an instruction cache, both with a size of 16 KByte. The hardware contains a 256 KByte Level-2 cache located on-die [17]. The main memory consists of 256 MBytes of PC100-SDRAM.

As operating system serves a Debian Linux 3.1 [9] with kernel 2.4.27 [20], compiled with support for a virtual floating point unit. Performance data for MIRA's hardware pattern database were determined with the *lmbench* testsuite [5]. For architectural exploration, the following hardware alternatives have been generated from the standard configuration: a version without Level-2 cache, without cache and without the hardware floating point unit. Every configuration was also modeled and converted to an Arena model.

For more accurate results, we sized the test programs with their inherent iterator and averaged the resulting running time. As compiler served the GNU C compiler, version 3.3.5-3. As application examples, we have chosen a compute- and an IO-intensive application.

As first application served the Fast Fourier Transform Benchmark *ffbench*, introduced in 1989 by John Walker [33]. This benchmark initializes a 256 by 256 array of double precision complex numbers to known values then executes twenty iterations through a loop, each iteration computing the two dimensional Fourier transform of the data in the matrix, then applying the inverse transform to recover the original data [32].

Table 2 shows the execution time for the *ffbench*, averaged to one iteration, on every architecture alternative and the associated simulation result.

Table 2. Comparison Between the Real and the Simulated Execution Time for the ffbench

configuration	real time in s	simulated time in s
Normal	0.44	0.45
Without FPU	19.36	19.34
Without L2 Cache	0.55	0.58
Without Cache	26.88	26.67

The second application exhibits IO behavior. This program reads a line with two numbers from a previous generated file and applies to the data four additions, two multiplications and a single division. We measured and simulated this behavior with float numbers and integer numbers. The fitting results, averaged to one million lines, are represented in Table 3.

The tables show, that the results obtained with MIRA and the measured execution-times are almost identical. With the proposed framework, users can build and simulate hardware / software systems to estimate their behavior.

5. Conclusion and Outlook

In this paper, we presented a method to generate EQN based simulation models for related design approaches. We

introduced the MIRA Framework, that enables us to build a system model using the proposed method. We showed an exemplaric EQN based model that represents a real life example.

Table 3. Comparison Between the Real and the Simulated Execution Time for Experiment 2

configuration	int/ float	real time in s	simulated time in s
Normal	int	2.00	2.01
Normal	float	2.01	2.02
Without FPU	int	2.00	2.01
Without FPU	float	10.56	10.11
Without L2 Cache	int	2.03	2.05
Without L2 Cache	float	2.04	2.06
Without Cache	int	514	512.84
Without Cache	float	515	513.73

Currently, we investigate the modeling capability for more complex computer systems with simultaneous execution of several applications. For that purpose, the utilization of an evaluation board, equipped with a CPU an an FPGA is intended. This will give us a larger range of worker units for evaluation. It will also allow the comparison of performance variables, like bus load or worker utilization, with measured values.

We also plan to introduce XML Schema definitions [31] for the verification of the XML representations, that are part of the MIRA Framework. We are confident, that XML Schema definitions enable us to do a syntactical as well as a semantical model verification with an off the shelf XML validator like Validome.

In further work, we will combine our model with the results of the works from Musovic et al. [21]. We expect, that this will enable us not only to predict performance constrains, but also power consumption of the intended computer system.

Future work will also include the development of a tool supported method to compare the results of the architectural exploration.

References

[1] Arena. Arena simulation software from rockwell automation. http://www.arenasimulation.com/, August 2006.

[2] S. Balsamo, A. Di Marco, P. Inverardi, and M. Simeoni. Model-based performance prediction in software development: a survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, May 2004.

[3] S. Balsamo, M. Marzolla, and R. Mirandola. Efficient performance models in component-based software engineering. In *SEAA '06 Proceedings of the 32nd Euromicro Conference on Software Engineering and Advanced Applications*, Cavtat/Dubrovnik, Croatia, 2006.

[4] F. Baskett, K. M. Chandy, R. R. Muntz, and F. G. Palacios. Open, closed, and mixed networks of queues with different classes of customers. *J. ACM*, 22:248–260, 1975.

[5] Bitmover. Lmbench - tools for performance analysis. http://www.bitmover.com/lmbench/, June 1998.

[6] Cadence. *Comprehensive VCC: Function / Architecture CO-Design and Design Export*. Alta Group of Cadence Design Systems, June 2001.

[7] V. Cortellessa, A. D'Ambrogio, and G. Iazeolla. Automatic derivation of software performance models from case documents. *Performance Evaluation*, 45(2-3):81–105, 2001.

[8] V. Cortellessa and R. Mirandola. Deriving a queueing network based performance model from uml diagrams. In *WOSP '00: Proceedings of the 2nd international workshop on Software and performance*, pages 58–70, New York, NY, USA, 2000. ACM Press.

[9] Debian. Debian linux homepage. http://www.debian.org/, 12 2006.

[10] P. Garg, A. Gupta, and J. Rozenblit. Performance analysis of embedded systems in the virtual component co-design environment. In *11th IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, 2004. Proceedings*, pages 61 – 68, May 2004.

[11] H. Gomaa and D. A. Menascé. Design and performance modeling of component interconnection patterns for distributed software architectures. In *WOSP '00: Proceedings of the 2nd international workshop on software and performance*, pages 117–126, New York, NY, USA, 2000. ACM Press.

[12] B. Gröne, A. Knopfel, and P. Tabeling. Component vs. component: why we need more than one definition [system component and software component]. In *12th IEEE International Conference and Workshops on the Engineering of Computer-Based Systems, 2005. ECBS '05*, pages 550 – 552, April 2005.

[13] G. P. Gu and D. C. Petriu. Xslt transformation from uml models to lqn performance models. In *WOSP '02: Proceedings of the 3rd international workshop on software and performance*, pages 227–234, New York, NY, USA, 2002. ACM Press.

[14] G. P. Gu and D. C. Petriu. Early evaluation of software performance based on the uml performance profile. In *CASCON '03: Proceedings of the 2003 conference of the Centre for Advanced Studies on Collaborative research*, pages 66–79. IBM Press, 2003.

[15] G. P. Gu and D. C. Petriu. From uml to lqn by xml algebra-based model transformations. In *WOSP '05: Proceedings of the 5th international workshop on software and performance*, pages 99–110, New York, NY, USA, 2005. ACM Press.

[16] IBM. A simple queueing network model of a time sharing system. http://domino.watson.ibm.com/comm/-research.nsf/pages/r.performance.history.html, May 2005.

[17] Intel. Intel pentium iii processor technical documentation. http://www.intel.com/design/archives/processors/-PentiumIII/index.htm, November 2006.

[18] C. Kabajunga and R. Pooley. Simulating uml sequence diagrams. In *Performance Engineering Workshop, UK PEW 1998*, pages 198–207, 07 1998.

[19] P. Kähkipuro. Uml-based performance modeling framework for component-based distributed systems. In *Performance Engineering - State of the Art and Current Trends, LNCS 2047*, pages 167–184, Berlin, 2001. Springer-Verlag.

[20] Linux. The linux kernel archive. http://www.kernel.org, November 2006.

[21] A. Musovic, R. Fischer, U. Nageldinger, K. Buchenrieder, and A. Lehmann. An eqn* based approach for high-level performance and power estimation. In *Conceptual Modeling and Simulation Conference, 2005. CMS 2005. 2nd 13M International Mediterranean Modeling Multiconference*, pages 193–198, 2005.

[22] OMG. The unified modeling language (uml), version 2.0. http://www.omg.org/technology/documents/formal/-uml.htm, July 2005.

[23] D. Petriu and M. Woodside. Analysing software requirements specifications for performance. In *WOSP '02: Proceedings of the 3rd international workshop on Software and performance*, pages 1–9, New York, NY, USA, 2002. ACM Press.

[24] D. C. Petriu and X. Wang. Deriving software performance models from architectural patterns by graph transformations. In *TAGT'98: Selected papers from the 6th International Workshop on Theory and Application of Graph Transformations*, pages 475–488, London, UK, 2000. Springer-Verlag.

[25] C. H. Sauer. Approximate solution of queueing networks with simultaneous resource possession. *IBM Journal of Research and Development*, 25(6):894–903, 1981.

[26] C. H. Sauer, E. A. MacNair, and S. Salza. A language for extended queuing network models. *IBM Journal of Research and Development*, 24(6):747–755, 1980.

[27] C. U. Smith. The evolution of software performance engineering: a survey. In *ACM '86: Proceedings of 1986 ACM Fall joint computer conference*, pages 778–783, Los Alamitos, CA, USA, 1986. IEEE Computer Society Press.

[28] Umbrello. Homepage of the umbrello uml modelling tool. http://uml.sourceforge.net/index.php, October 2006.

[29] Validome. Validator for xml documents from validome. http://www.validome.org/xml/, August 2006.

[30] W3C. W3c site of the extensible markup language (xml). http://www.w3.org/XML/, September 2006.

[31] W3C. Xml schema. http://www.w3.org/XML/Schema, October 2006.

[32] J. Walker. Homepage of fast fourier transform benchmark. http://www.fourmilab.ch/fbench/ffbench.html, November 2004.

[33] J. Walker. Homepage of floating point benchmarks. http://www.fourmilab.ch/fbench/, August 2005.

[34] M. Woodside, D. C. Petriu, D. B. Petriu, H. Shen, T. Israr, and J. Merseguer. Performance by unified model analysis (puma). In *WOSP '05: Proceedings of the 5th international workshop on Software and performance*, pages 1–12, New York, NY, USA, 2005. ACM Press.

[35] Z. Xu and A. Lehmann. Automated generation of queuing network model from uml-based software models with performance annotations. Technical Report 2002-06, Universität der Bundeswehr München, Fakultät für Informatik, Munich, Germany, September 2002.

[36] Z. Xu and A. Lehmann. Extending functional models for predicting software performance during the unified software development process. In *WPES' 03: Proceedings of the 4th Workshop Performance Engineering in der Softwareentwicklung*, Magdeburg, Germany, 2003.

[37] Z. Xu and A. Lehmann. Generating queuing network models from uml-based models for software performance prediction. In *SPECTS' 04: Proceedings of the 2004 International Symposium on Performance Evaluation of Computer and Telecommunication Systems*, San Jose, California, USA, 2004.

Formal architecture transformation using heuristics

Cameron Maxwell[1,2], Tim O'Neill[2], John Leaney[2]
1. Faculty of Information Technology
2. Institute for Information and Communication Technologies
University of Technology, Sydney
cmaxwell@it.uts.edu.au
[Tim.ONeill | John.Leaney]@uts.edu.au

Abstract

Heuristics have long been a popular and effective mechanism for capturing the knowledge of experts. In recent times, however, the more common use of heuristics has been as a means for communicating ideas at an abstract level, with little consideration to their potential as a structured approach to design improvement. With this paper we present the issues surrounding, and a structured method for, formally capturing architectural change embodied within heuristics. We demonstrate how through the application of graph theory, category theory and predicate calculus we can capture change within a heuristic and then use it to achieve formal heuristic-based transformation of a real-world system. By capturing heuristics in the structured and formal manner discussed in this paper we present ourselves with the opportunity to create a practical and reliable heuristic-based architecture transformation system. This is done within the wider context of achieving a process for optimising the non-functional qualities of a system architecture through design transformation.

1. Introduction

Managing design conflicts is a principal role for the designers of today's modern complex computer-based systems. In particular, they are often responsible for ensuring that the design and architecture of a system not only fulfils its functional requirements but its non-functional ones as well. This becomes a significant problem for designers as complex systems often have many competing requirements and more often than not too many for the designer to consider at once.

In response to this researchers have produced a number of methods for modelling and analysing system designs and architectures. These aid designers in understanding the effects of architectural design choices on systems as a whole and their flow-on effects to system requirements. We believe the next logical step from here is to develop methodologies and tools that allow system designers to optimise their designs, as is common with virtually all existing engineering and design disciplines.

In the context of architectural design we posit that a degree of optimisation can be achieved by reducing, or minimising, the conflicts between competing requirements. Specifically, as the analysis of the system's architecture is considered to be fundamental to ensuring the non-functional qualities of a system, by minimising the conflicts between these qualities we should be able to produce systems that better, or more optimally, achieve all their non-functional requirements.

In this paper we will be focusing on one aspect of such an optimisation process, the design change process. Design change, or manipulation, is the process by which we elicit change in a system's architecture (design) and hopefully move its design towards a more optimal state. Specifically, we propose that capturing expert knowledge in the form of heuristics is a part of valid approach to achieving this aim. The overall context in which heuristics help to achieve an optimal system is described in the next section.

2. Framework overview

Over the past five years we have been developing a framework that would allow us to achieve optimisation by minimising conflicts. After an initial proposal and exploration of the problem space [18] we have developed an optimisation framework for optimising non-functional qualities in computer-based systems. Figure 1 gives a high-level view of this framework.

This framework, originally modelled on a control system-like feedback loop, breaks up the necessary steps required to optimise an architecture into logical, discrete, components. Under this framework the following iterative process is applied:

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

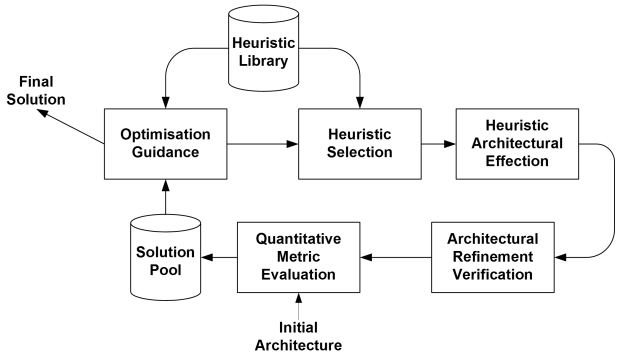


Figure 1. Architecture-based design optimisation framework

1. The first step is to define goals for the optimisation process. These are, which non-functional qualities do we wish to reduce the conflicts between, and what minimal levels need to be achieved in these qualities for the design to be considered valid.
2. An initial architecture is evaluated to identify its current non-function qualities. These are measured by metrics that can either be calculated directly from the design, or implied via simulation. This architecture and its metrics provide the baseline for the optimisation process and both are placed into the solution pool.
3. The guidance mechanism examines all architectures currently in the solution pool and their metric values to find out if any of the architectures meet the goals for the optimisation. If they do not, an architecture is selected which the guidance mechanism believes could be improved and moves that design closer to goals.
4. A selection process is conducted to find a heuristic from a prepared heuristic library that has the potential to improve the selected architecture. Each heuristic embodies an architectural design change that is intended to improve a given non-functional quality of the system.
5. The *Heuristic effection* then occurs. This takes the selected architecture and the selected heuristic and applies the changed described within it. This occurs using formal transformation methods and hopefully produces an "improved" architecture.
6. Architectural refinement and verification then occurs to ensure that during the transformation process the architecture has not been modified in a way that would prevent it from meeting its functional requirements.
7. Finally the new architecture is evaluated, by calculating for the new architecture all of the desired non-functional qualities, to determine if it is in fact better than the original, or if the change has in fact degraded the architecture. Regardless, the new architecture is placed into the solution pool ready for the next iteration.

As should be evident, this framework utilises a library of heuristics, that is ways in which it can change an architecture to improve it. In our case we have chosen to use heuristics to capture these changes as they provide an effective solution for sourcing changes. By using heuristics we can capture the expert knowledge of system designers who are already familiar with manipulating the designs of complex systems. Furthermore, this approach, by utilising changes known to have worked for experts in the past, reduces the risk of the optimisation process producing invalid designs.

For heuristics to be stored and then subsequently used to effect design change, as required by this framework, a formal representation is required. The formal representation must provide a comprehensive mechanism that preserves the fundamental concept of a heuristic whilst allowing for repeatable and reliable changes to be effected on the design. The heuristic representation and transformation mechanism presented in this paper are intended to fulfil these requirements.

3. Background

Heuristics are not a new concept. In one form or another they have existed for over two thousand years. In simple terms, a heuristic is:

> *A commonsense rule (or set of rules) intended to increase the probability of solving some problem.*[1]

Heuristics differ from other design support methods, such as design patterns, by the context and hierarchy under which they apply. Design patterns, for example, are intended to provide very specific solutions at a very specific point the design hierarchy. Heuristics on the other hand tend to be more general statements that can be applicable at numerous levels within a design hierarchy.

In effect, heuristics capture ways we believe we can solve some problem. What they do not do, however, is provide any guarantee that they will solve anything, only that they may do so. Due to this heuristics are often given very broad application conditions to give them a greater opportunity of having a positive effect. Both these properties of heuristics are crucial to their effectiveness and need to be embodied within any representation.

Interest in the use of heuristics as a problem solving mechanism has undergone a significant revival in the last fifty years. Computer scientists and engineers alike have sought to use them to explain problem solving approaches

and in capturing human knowledge. Koen, in a more philosophical look at heuristics, likens them to the basis for the entirety of the "engineering method" [14].

One of the first modern uses of heuristics was in the development of inference engines. Of these the MYCIN [26] expert system is by far the most well known. The heuristics in MYCIN were represented by relatively simple *if-then-else* constructions. MYCIN combined these heuristics to produce complex reasoning and inference. Although we are not seeking to create a complex reasoning framework, the simple structure of the MYCIN heuristics provides a good basis from which we can begin to describe our own heuristics. Despite the technical success of MYCIN heuristics quickly fell out of out of favour in the artificial intelligence domain due to the perceived problems associated with their inherent fallibility [24]. We believe, however, that it is their inherent fallibility, that is the fact the heuristics may not always provide a correct answer, which makes them ideal for our purposes.

Despite the use of heuristics for millennia as a means for guiding problem solving, there has been a growing interest in recent times to understand their use in the context of modern engineering practices. This work has varied from highly philosophical views of problem solving and the engineering process [14] to more concrete attempts to explain heuristics place in systems and computer engineering [23]. Unfortunately, as heuristics are at their most powerful when used to describe abstract concepts, little work has been done to codify or to provide a method for formally describing heuristics. This work present in this paper hopes to begin to rectify this.

For heuristics to be able to optimise an architecture they must be able to change, or transform, its design. Architecture transformation, however, is inherently dependent on a valid architectural model that can be used to represent the architectures that are to be transformed. Extensive research has been conducted into many different ways to model architectures, both formally and informally. Early work by Perry and Wolf [22], Shaw [25], Garland [11, 12] and Allen [2] serve to provide us with what are the essential basics of any model. For model transformations a vast number of approaches are available to us. A good taxonomy of model transformation methods can be found in [5].

A common method for not only modelling architectures but transforming them is through the use of graphs. For this reason there is a great deal of existing research on using graphs in architecture. Graphs have been used to model style [10, 16], to perform verification [15] and for cross-domain translation [27]. All these applications of graph theory, and graph grammars in particular, to modelling architecture serve to demonstrate the flexibility of graphs to modelling architectures. Further, their success also serves to demonstrate the appropriateness of using graphs to model the structures present in systems architectures [6, 7].

Graph-based representation are especially powerful when they are used in cooperation with category theory. Specifically category theory has a construction known as a *pushout* which can be used to provide repeatable, deterministic formal graph transformations [8]. Pushout based graph transformations have already been used in other architecture transformation systems with some success. Wermelinger and Fiadeiro used pushouts to construct an architecture transformation system for dealing with dynamic reconfiguration [28]. Due to the focus on dynamic reconfiguration their work is heavily concentrated on state preservation during a transformation, an issue which does not effect our approach as we are only interested in static design transformations.

4. Heuristic model

In our previous examination of heuristics [17], we identified a framework for deriving a descriptive model of heuristics used in architectural change. This framework showed how a heuristic could be broken down into its constituent parts. Accordingly, a heuristic consists of the following:

- A description of the *goal* of the heuristic. That is, which non-functional property of the system is the heuristic seeking to improve.
- The expected *benefit* of the heuristic. This is the expected quantitative change in the non-functional goal quality.
- The *constraint* of the change. The constraint is what must exist in the system architecture before the heuristic can be applied.
- The *change* itself, a description of how the design of a system changes to implement the heuristic.

Each of these elements plays a specific role in how a heuristic is to be applied. The *goal* and *benefit* assist in determining which is the correct heuristic to use in any situation by describing its intended purpose. That is the non-functional quality it is intend to improve, as well as how significant an effect it might have. The *constraint* and *change* then describe where and how the architecture will be changed by the heuristic.

This framework serves as the basis for the construction of a formal model of heuristics. Taking this framework we use a combination of predicate logic and graph theory as the grounding formalism from which to create the formal description of a heuristic. In doing so, to create a complete description of a heuristic, we must create a formal description for each of the four elements that make up a heuristic. The *goal*, *benefit*, *constraint* and *change*.

Before describing each of these elements it is important to point out that their selection and definition are such

that as to ensure that the special qualities of heuristics are preserved. We previously mentioned that being fallible and possessing a general application condition were inherent properties of heuristics and it is the general application condition that must be ensured in the formal representation.

We achieve this generality of application condition in two ways. The first is a general guideline to be followed when specifying a heuristic. The guideline states that "*Each heuristic should be given the minimal possible constraints.*" This guideline helps to remind and ensure that when capturing heuristics, they must be kept as generic as possible by not limiting the conditions under which they can be applied. The second method we use for ensuring the generality of the application condition is a function of the way in which we describe the goals of a heuristic. We will describe this in more detail in the following section.

4.1. Modelling goals

The first element of our heuristic that we need to model formally is the goal of the heuristic. In our case this is very straight forward. As we are using our heuristics to improve system architectures, specifically the non-functional qualities of those architectures, then the goals of our heuristics are those non-functional qualities the heuristic effects. To make these goals both formal and practical, however, we need to refine this further.

Non-functional qualities are often talked about in terms of their generic names such as *reliability*, *security*, *maintainability*, etc. Unfortunately, these terms are often poorly defined and have a different meaning depending on the context of their use. Their meaning can even change from system to system [21]. This renders these generic non-functional qualities unusable in describing the goals for our heuristics. The reason for this is that the ultimate use of our heuristics is to provide a mechanism for changing an architecture within an optimisation process. For any optimisation process to work you must be able to measure and evaluate those properties which you are trying to optimise. Without a standard definition for non-functional qualities this becomes impossible.

Fortunately, there is a simple and highly effective mechanism by which we can define formal and measurable goals for heuristics whilst still aligning them with non-functional system qualities. In reality when we try to measure non-functional qualities what we are actually doing is collecting metrics and then mapping these metrics onto non-functional qualities. Effectively this makes non-functional qualities a composite measure of the metrics within the system. Take for example the metrics *mean time between failure* (MTBF) and *mean time to repair* (MTTR). These metrics can be combined into a single measurement that can be used to describe *reliability* and *maintainability* respectively. There is not, however, always an obvious mapping from metrics to non-functional qualities. In the case of *maintainability*, MTBF and MTTR may not be the only factors that determine how "maintainable" a system is. Determining the maintainability of the system may also need to consider the cost involved in the maintenance. A system may in fact be far less "maintainable" if the time to repair the average failure is very short but the cost of doing so exceeds the budgetary constraints under which the system must operate.

What this means for our formal model of heuristics is that if we use metrics as the basis of our formal model of a heuristics goal we gain some very distinct advantages. Firstly, defining metric based non-functional qualities for the goal gives us a common agreed upon and measurable "property" that should not change its meaning from system to system. Secondly, by allowing metrics to be mapped onto different non-functional qualities as they relate to the system we create a very powerful mechanism under which heuristics can influence a varied number of non-functional qualities without any need for modification or re-specification.

It is this second point that also helps to ensure that we are maintaining the generality of the application condition required by heuristics. As metrics may have flow-on effects to a number of non-functional qualities, expressing the goal of a heuristic in terms of a metric gives it the potential for a wider range of effect. Furthermore, by limiting the goal of a heuristic to a simple identification of metrics we eliminate the risk that other more complex goal specifications would put too stringent a condition on where the heuristic could be applied.

Finally, by using metrics to define the goal of a heuristic we gain a very simple and easy mechanism by which to categorise it. By categorising heuristics by the metrics they are believed to effect, they can easily be found and selected by the guidance mechanism of the proposed optimisation framework when it is looking to optimise a specific non-functional quality.

Given all this, our formal model for the goal of the heuristic is a list of metrics. The metrics in this list are those metrics which the heuristic is expected to give a "positive" improvement for that metric. Positive improvement is defined as movement in the metrics value towards an ideal value, thus for metrics with "lower is better" values, such as MTTR, positive improvement means a negative or lower value for the metric.

4.2. Modelling benefits

Defining the benefit of a heuristic, namely an expected measure of what level of improvement the heuristic should have on its goal metrics, is a difficult task. This difficulty is exacerbated by the very nature of heuristics.

Heuristics by their very nature are often simply statements of an action with a positive outcome. As such they often contain little or no information as to what degree of improvement can be expected through their application. Despite this there is a very important reason for wanting to include benefit details in a formal model of heuristics.

Defining a benefit of a heuristic allows us to differentiate between heuristics with the same or similar goals. Being able to differentiate between heuristics in this manner is critical to their use in an optimisation process. Consider the situation where a system has been under optimisation for some time already and the only quality left that has not reached its optimised target is reliability. Let's assume that we have two heuristics that purport to improve reliability, one by 50%, the other by 10%. If for the sake of this demonstration we assume the following (although these assumptions may not hold for all heuristics, as it is possible that they may hold for some it is important to consider them):

- A heuristic with a greater benefit implies a greater change to an architecture; and
- The greater the change to an architecture the greater the likelihood that it will effect other metrics, potentially negatively.

Then it follows that the greater the benefit provided by a heuristic, the greater the likelihood that it will negatively effect other metrics or qualities. In this situation, where the system has already been optimised to a high degree it becomes highly desirable to apply those heuristics that will provide a benefit without disturbing or compromising the optimisation effort already undertaken. It is for this reason that it is important our heuristic model provides a mechanism to differentiate heuristics based on their expected benefit.

Formally, we define the expected benefit of a heuristic in two ways. We state that the expected benefit of a heuristic may either be *relative* or *absolute* improvement. Relative improvement indicates that the benefit arrived from applying the heuristic is in some way "relative" to the system that existed before it. For example a heuristic that places a second server in parallel with an existing server for load balancing could claim that it expected to reduce processor load by 50%.

Conversely *absolute* improvements indicate that the expected benefit is fixed. This type of expected benefit is often a reflection of a "replacement" type change, where one element in an architecture is swapped for another with different properties. One example might be a heuristic that specifically removes one type, or brand, of server and replaces it with another type where the difference in MTBF between both types is known.

4.3. Modelling constraints and changes

When considering the constraints and changes captured within a heuristic it is helpful to examine them together as their use is tightly coupled. The constraint of a heuristic identifies what must be present in an architecture before it can be applied, whilst the change describes what manipulations the heuristic makes to an architecture in order to try and achieve its goal.

In a heuristic both the constraints and the changes may be classified into either *property* or *structural* constraints and changes. Structural constraints describe what architectural structure must be present before the heuristic can be applied. Whilst the property constraints describe what properties those architectural elements present in the structural constraint must have. Similarly structural changes describe the way in which the heuristic changes the structure of an architecture and property changes describe the way the properties of the elements within the architecture change.

These similarities allow for us to vastly simplify the treatment of constraints and changes. Furthermore, there are enough similarities for us to use a common method of representation for both the constraint and the change.

Formally, we represent both as an architectural model or slice. The constraint slice contains the model of what architectural structure, including its associated properties, must exist before the heuristic can be applied. The change slice represents what the constraint slice should be after the heuristic has been applied. Essentially, the constraint of a heuristic has become the pre-condition of its application whilst the change description is captured through the combination of the pre- and post-condition. Each of these slices can be described by a formal model of an architecture, and in fact the same formal model may be used for both. The model we use to describe these is outlined in the next section.

5. Formal architecture model

As already discussed a common architecture model provides for a simplified description of the constraint and change elements of the heuristic. Producing a formal model flexible enough to describe all possible heuristics, however, requires that our model be able to model an architecture in full.

As such we began by identifying which elements, and the relations between those elements, we would be required to model. Using the commonly accepted notion of an architecture put forward by [19, 20], we have identified the following architectural elements:

- Component
- Connection

- Port

We also have the following relationships between these elements:

- Parent-child relationship; and
- Port-to-Port bindings.

We are able to capture all of these elements and relations using graphs. The nodes of a graph are used to represent the architectural elements (components, connections and ports) and the edges represent the relationships between the elements (parent-child and port-to-port). This simple graph representation captures the structure of an architecture, however, it is not complete as it fails to capture the properties of those elements that make up the architecture. e.g. The cost of components, the speed of network links, etc. To capture these properties we need to use an extension of graph theory that allows us to assign properties, or attributes, to the nodes and elements in the graph. There are a number of existing methods available to do this [9]. With the inclusion of the element properties we now have a representation with which is it possible to represent not only an architecture but also the constraints and changes of heuristics.

Although a full description of the architectural model is beyond the scope of this paper, the following is an introduction to the basic structure of the model. Formally, we state the model of an architecture A is fully described by the following:

$$A = (\Psi_A, \Upsilon_A, G_A) \tag{1}$$

Where:

$A =$ The architecture model
$\Psi_A =$ The set of property relations P for graph nodes
$\Upsilon_A =$ The set of property relations for graph edges
$G_A =$ The graph with the architectural structure

All architectural elements within an architecture, that is the components, connections and ports, may have properties associated with them. Each element has a *property relation* (2) which captures these properties as name and value pairs. The set of these relations for all elements in the architecture model is Ψ_A.

$$P : n \nrightarrow v \tag{2}$$

The relationships in an architecture model do not have properties in the same sense as architecture elements. It is, however, useful and desirable to allow certain "properties" to be specified for these. We use the same property relation (2) to store their properties and Υ_A is the set of all these property relations.

The graph (3) stores the structural model of the architecture. The vertices, or nodes, representing the architectural elements are V_{G_A}, with the graph's edges being E_{G_A}. The mapping of edges to their source and target nodes is achieved through (4).

$$G_A = (V_{G_A}, E_{G_A}, s_{G_A}, t_{G_A}, \psi_{G_A}, \upsilon_{G_A}) \tag{3}$$
$$s_{G_A}, t_{G_A} : E_{G_A} \rightarrow V_{G_A} \tag{4}$$

Finally we include mappings from the graph's nodes to their associated property relations (5), as well as from the graph's edges to their property relations (6).

$$\psi_{G_A} : V_{G_A} \rightarrow \Psi_A \tag{5}$$
$$\upsilon_{G_A} : E_{G_A} \rightarrow \Upsilon_A \tag{6}$$

6. Example

To demonstrate how representing heuristics in this manner can work we will use an example taken from the real world. We present the example as the specific version of a more general change. Although this may seem counter intuitive to the notion of a heuristic it greatly simplifies the example and a full example of general heuristic specification is beyond the scope of this paper.

The following example is taken from a real system and the heuristic was derived from existing knowledge concerning system development. For privacy reasons identifying details have been removed.

The aim of the *ACME Training Project* was to integrate a number of different training systems located in a range of different locations. The most significant problem facing the project was that the systems to be integrated came from a range of different pedigrees and indeed utilised a number of different data formats and protocols. A survey conducted at the start of the project identified 20 systems, located in 8 sites and that used 5 different protocols to communicate.

An examination of the different protocols used by the systems found that they were amenable to translation between one another. Consequently it was decided to employ a number of *translation gateways* to convert between the protocols. The original design for the project called for these gateways to be present at every site to translate the communications before being sent over the wider network.

This use of many separate gateway resources is a reflection of a common problem facing distributed system design. That is "What is the correct balance to achieve between centralised and decentralised allocation of resources and infrastructure?" This problem arises in a number of domains including network design and management[4], distributed processing [13] and security [3]. As a common problem

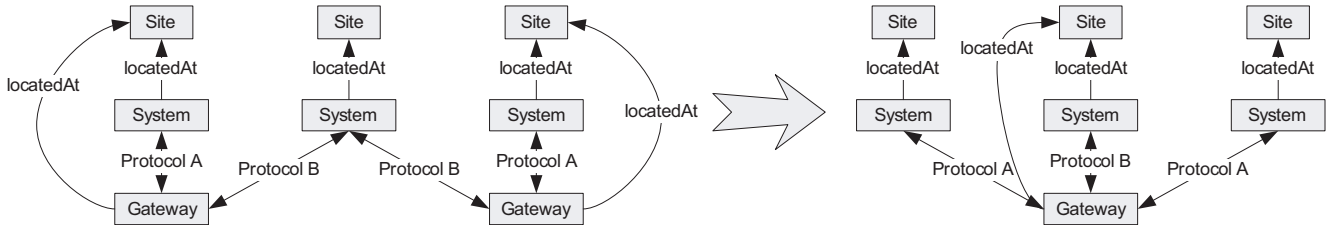


Figure 2. Heuristic summary - centralise gateways

there exists some knowledge that we take from these domains and capture as heuristics.

For our example we will focus on the concept that centrally located resources reduce costs. Our first step is to state this concept, or "heuristic", in terms of the domain of our problem. In the case of the *ACME Training Project* we can summarise the heuristic as follows:

> *The cost of a system can be reduced by replacing separate distributed translation gateways with a common, centralised one.*

Figure 2 gives a graphical summary of how we might expect this sort of transformation to be represented in the structural model of a system.

Using these summaries as a starting point we can now proceed to specify each of the elements of the heuristic formally. As identified in Section 4 we need to specify:

- the *Goal*
- the *Benefit*
- the *Constraint*, and
- the *Change*.

6.1. Goal

The first element specified formally is the goal of the heuristic. In practice when determining the goal of a heuristic, or the benefit for that matter, often these will simply be a statement of opinion. This is because the heuristics will be captured from experts and it is their intent and rationale for using a heuristic that we are trying to capture. That said, there is a possibility in the future that the metrics which a heuristic could effect may be able to be determined by examining the change itself.

In the case of our example it is clear that from the statement of the heuristic above we hope to improve the cost of the system. In reality, many heuristics can be expected to effect more than one metric and it is useful to try and express them all. However, to simplify this example we will only consider cost. So formally we specify the goal as "cost".

$$\text{Goal} = \{\text{Cost}\} \tag{7}$$

6.2. Constraint

To formally capture the constraint of the heuristic we have to capture its pre-conditions. In this case looking at Figure 2 we can see that the left-hand side of the figure represents the pre-condition of the change. It contains the elements we expect to exist within an architecture before we would apply the change, and how they should be connected. Using this summary as a base we can enrich its description to meet our formal needs.

We begin by modelling the structure of the architecture that will serve as the constraint and assigning identifiers to all the elements. Figure 3 shows the beginning of this process (identifiers have been left off the ports for brevity). On a formal level each component, connection and port is represented by a node on the graph, however, we use different shapes for each to aid the communication of heuristics.

With a little examination it can be seen that nodes $\boxed{1}$, $\boxed{4}$ and $\boxed{6}$ reference the "sites", $\boxed{2}$, $\boxed{5}$ and $\boxed{7}$ are the "systems", and finally $\boxed{3}$ and $\boxed{8}$ are the gateways. This type information, as well as the other properties of the elements is formally represented as a property list. For example $\boxed{3}$ would contain a property list such as the following:

$$\{\text{id} \rightarrow 3, \text{type} \rightarrow \text{“Gateway”}, \text{cost} \rightarrow \$49000\} \tag{8}$$

6.3. Change

As stated previously, the change of a heuristic is captured by describing the pre- and post-conditions of the transformation. As we already have the pre-conditions described as the constraint (Figure 3), all that remains is to describe the post-conditions.

The simplest way to go about constructing the post-condition graph for a heuristic is to start by taking a copy of the constraint graph. Using this copy we manipulate the

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

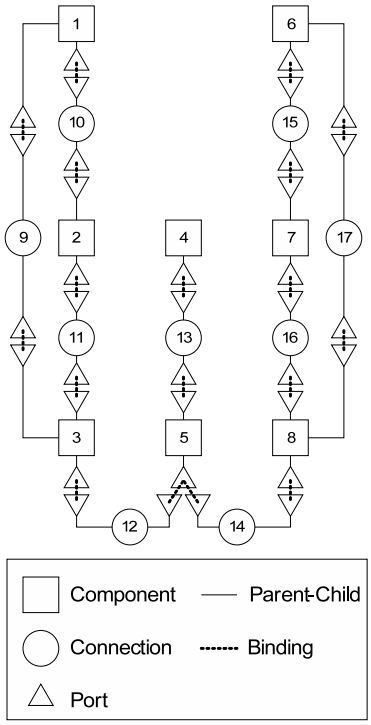


Figure 3. Formal structural constraint

graph to show what the graph, and by proxy the architecture, would be if the heuristic was applied to the constraint graph.

Figure 4 shows the post-condition graph for our example heuristic. Note that elements with the same identifier map between the pre- and post-condition graphs. This is critical in determining which elements may have changed during the transformation.

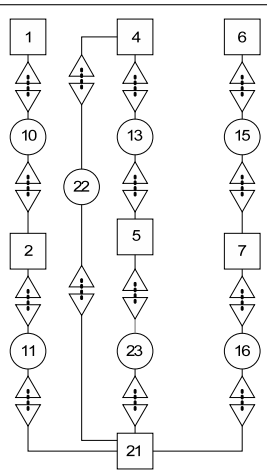


Figure 4. Formal structural change

It can clearly be seen how this graph was constructed from the initial pre-condition graph and then modified. The new elements added to this graph can be identified as they have been labelled with the numbers 21, 22 and 23. Further it should be clear how this post-condition graph reflects the change we are trying to elicit, as summarised in Figure 2.

6.4. Benefit

Estimating the benefit provided by this heuristic, that is what reduction in cost can be expected, is a difficult task. Primarily this is due to the many different factors that can effect it. We don't, however, need a perfect prediction of the expected benefit but only an estimate. Which means we can use a much simpler measure of the expected cost savings.

We do this by specifying the expected benefit in terms of the change being applied after the constraint is matched. That is, we use the constraint or pre-condition of the heuristic to tell us what the expected benefit will be. This is why we have left specifying the expected benefit till last.

If we drastically simplify our heuristic to the point where we consider it as replacing two gateways with one it becomes obvious that the cost saving implied is the cost of one gateway. Referring to Figure 3 we can pick either of the gateways being removed $\boxed{3}$ or $\boxed{8}$ to be the basis of our benefit estimation. Letting $\boxed{3}$ be $gateway$ we can state:

$$\text{Benefit} = propertyValue(gateway, \text{“cost”}) \quad (9)$$

Where:

$$v = propertyValue(x, n) \Leftrightarrow P_x : n \mapsto v \quad (10)$$

This expresses the expected benefit, the reduction in cost, of applying the heuristic is directly related to the properties of $\boxed{3}$. Or more accurately, the cost reduction will be equal to the cost of the gateway in the architecture to which $\boxed{3}$ maps to when the heuristic is applied.

6.5. Applying the heuristic

Now that we have a complete formal description of our heuristic we can apply it to the architecture of the *ACME Training System*. To do this we take a model of the system architecture and apply the heuristic wherever possible.

As the task of searching a graph for matching conditions to identify where a heuristic may be applied is not a simple issue we use a tool designed for this very purpose. AGG [9] is a tool designed for performing graph transformation on *attributed graph grammars*, and uses a *single pushout* (SPO) internally, making it is ideal for our purposes.

Using AGG a graph was created to model a simplified version of the *ACME Training Project* system architecture. We simplified the model to only include details of the

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

locations, systems and gateways present. This simplification is necessary as the size of the graphs can be very large and very time consuming to process. Even after simplifying the model the final graph still consisted of approximately 400 nodes and edges. With the model in place we iteratively applied the heuristic transformation and examined the results.

The first iteration found a match for the pre-condition of the heuristic and applied it to centralise the gateways matched. An evaluation of the resulting architecture showed that the total system cost had been reduced by approximately $20,000. This naturally corresponds to the cost of the gateway that was removed. The second iteration matched gateways using a different protocol to the first iteration. When these were centralised a further saving of $35,000 was achieved. The increased saving was due to the increased cost of the second set of gateways which were more expensive due to the complexity required in their operation.

On the third iteration no further matches where found. An investigation as to why a match couldn't be found revealed an interesting fact about the system that had been initially overlooked. Although the initial audit identified a considerable number of different protocols and systems when we examined the system architecture in detail we found that particular locations tended towards using the same protocols. This was not, however, due to any intentional design, but rather a chance coincidence. As a result there were in fact only two locations that required different types of gateways, both of which were those identified by our heuristics.

Despite the limited number of iterations in which we were able to apply the heuristic to this system, we were still able to "improve" the design of the system with respect to its cost. Which was the aim of the example.

7. Further work

One of the outstanding issues that hasn't been finalised is how to support variable, or what we are calling "tuneable" heuristics. These heuristics have the characteristic that their application can be varied to "tune" the effect they have on an architecture. For example, a heuristic that creates additional processing components in parallel to reduce load could be "tuned" by specifying the number of additional components to add. This level of control would be hugely advantageous in an optimisation process.

Implementing this in a formal manner poses some distinct difficulties. Currently we are working with sub-graphs within the constraint and change specifications that we hope could potentially be replicated a given number of times to provide this ability.

Also, given the considerable size of the model and graphs involved we have begun work on a suitable mechanism for creating "short-hand architecture graphs". These graphs allow the architecture to be described succinctly and then transformed into the full formal graphs required for the optimisation process.

8. Conclusions

Heuristics have long been used to embody the capturing of human expertise and experience. In this paper we have put forward a notation by which they can be represented formally for use in optimising the non-functional qualities of system architectures. Furthermore, we have shown how the proposed notation can be used in the automatic transformation of architectural models to effect the change embodied within heuristics.

Using a framework consisting of goals, benefits, constraints and changes as a base we constructed a formal representation of heuristics. This representation uses graphs to model architectures and architectural structures. These graphs in turn are used to capture the pre- and post-conditions of the changes implied by the heuristic. Category theory then allows us to use these to effect the transformation of the architecture using single pushouts.

Finally, our example showed that a heuristic represented in this manner can be used to transform architectures in a valid, formal manner. It also demonstrated that these transformations, these applications of heuristics, were able to produce an improved design.

References

[1] *Wordnet 2.0.* Princeton University, 2003.

[2] R. Allen. *A Formal Approach to Software Architecture*. PhD thesis, Carnegie Mellon, School of Computer Science, January 1997. Issued as CMU Technical Report CMU-CS-97-144.

[3] J. C. N. Payne. Using composition & refinement to support security architecture trade-off analysis. In *Proceedings from the 22nd National Information Systems Security Conference*, 1999.

[4] T. M. Chen and S. S. Liu. A model and evaluation of distributed network management approaches. *IEEE Journal on Selected Areas in Communications*, 20(4):850–857, May 2002.

[5] K. Czarnecki and S. Helsen. Classification of model transformation approaches. In *Workshop on Generative Techniques in the Context of Model-Driven Architecture (OOPSLA?03)*, 2003.

[6] M. Denford, J. Leaney, and T. O'Neill. Non-functional refinement of computer based systems architecture. In *Engineering of Computer-Based Systems, 2004. Proceedings. 11th IEEE International Conference and Workshop on the*, 2004.

[7] M. Denford, A. Solomon, J. Leaney, and T. O'Neill. Architectural abstraction as transformation of poset labelled graphs. *Journal of Universal Computer Science*, 10(10):1408–1428, 2004.

[8] H. Ehrig, R. Heckel, M. Korff, M. Löwe, L. Ribeiro, and A. Wagner. *Handbook of graph grammers and computing by graph transformation*, volume 1: Foundations, chapter Single pushout approach and comparison with double pushout approach, pages 247–312. World Scientific Publishing, Singapore, 1997.

[9] C. Ermel, M. Rudolf, and G. Taentzer. *Handbook of graph grammers and computing by graph transformation*, volume 2: Applications, Languages and Tools, chapter The AGG approach: Language and environment, pages 551–604. World Scientific Publishing, Singapore, 1997.

[10] H. Fahmy, R. Holt, and S. Mancoridis. Repairing software style using graph grammars. In *Proceedings of the 1997 conference of the Centre for Advanced Studies on Collaborative research*, page 6. IBM Press, 1997.

[11] D. Garlan, R. Allen, and J. Ockerbloom. Exploiting style in architectural design environments. In *Proceedings of SIGSOFT '94: 2nd ACM SIGSOFT Symposium on the Foundations of Software Engineering*, pages 175–78. ACM Press, December 1994.

[12] D. Garlan, R. Monroe, and D. Wile. Acme: An architecture description interchange language. In *Proceedings of CASCON'97*, pages 169–183, November 1997.

[13] J. L. King. Centralized versus decentralized computing: organizational considerations and management options. *ACM Comput. Surv.*, 15(4):319–349, 1983.

[14] B. V. Koen. *Discussion of the method: Conducting the engineer's approach to problem solving*. Oxford University Press, 198 Madison Avenue, New York, New York, 2003.

[15] J. Kong, K. Zhang, and G. Song. A graph grammar approach to software architecture verification and transformation. In *27th Annual Interation Computer Software and Applications Conference 2003*, pages 492–497, 2003.

[16] D. Le Metayer. Describing software architecture styles using graph grammars. *IEEE Transactions on Software Engineering*, 24(7):521–533, 1998.

[17] C. Maxwell, T. O'Neill, and J. Leaney. A framework for understanding heuristics in architectural optimisation. In *13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2006)*, Potsdam, Germany, March 2006.

[18] C. Maxwell, A. Parakhine, M. Denford, J. Leaney, and T. O'Neill. Heuristic-based architecture generation for complex computer systems. In *12th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2005)*, 4th-7th April 2005.

[19] N. Medvidovic and R. N. Taylor. A classification and comparison framework for software architecture description languages. In *IEEE Transactions on Software Engineering*, volume 26, pages 70–93, 2000.

[20] T. O'Neill, J. Leaney, M. Denford, D. Colquitt, D. Turton, N. Sheridan-Smith, and J. Wootton. Current developments in the definition and description of system architectures. In *11th Annual IEEE International Conference and Workshop on the Engineering of Computer-Based Systems, Model-Based Development Workshop (ECBS2004)*, 2004.

[21] T. O'Neill, D. Rowe, and J. Leaney. An open computer based system (cbs) quality metrics framework. In *11th IEEE Conference and Workshop on Engineering of Computer-Based Systems (ECBS'98)*, Jerusalem, Israel, 1998.

[22] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *ACM SIGSOFT Software Engineering Notes*, 17(4):40–52, October 1992.

[23] E. Rechtin. *The Art of Systems Architecting*. CRC Press LLC, 2nd edition, 2000.

[24] S. J. Russell and P. Norvig, editors. *Articial Intelligence: A Modern Approach*. Prentice-Hall International, 1995.

[25] M. Shaw, R. DeLine, D. V. Klein, T. L. Ross, D. M. Young, and G. Zelesnik. Abstractions for software architecture and tools to support them. *Software Engineering*, 21(4):314–335, 1995.

[26] E. Shortliffe. *Computer-based medical consultation: MYCIN*. Elsevier, New York, 1976.

[27] J. Sprinkle, A. Agrawal, T. Levendovszky, F. Shi, and G. Karsai. Domain model translation using graph transformations. In *10th IEEE Internation Conference on the Engineering of Computer-Based Systems (ECBS2003)*, pages 159–168, 2003.

[28] M. Wermelinger and J. Fiadeiro. A graph transformation approach to software architecture reconfiguration. In *Joint APPLIGRAPH/GETGRATS Workshop on Graph Transformation Systems (GRATRA2000)*, pages 134–141, 2000.

A Case Study: Applying Lyra in Modeling S60 Camera Functionality

Jukka Honkola
Nokia Research Center
jukka.honkola@nokia.com

Sari Leppänen
Nokia Research Center
sari.leppanen@nokia.com

Pasi Rinne-Rahkola
Nokia Research Center
pasi.rinne-rahkola@nokia.com

Martti Söderlund*
TietoEnator
martti.soderlund@tietoenator.com

Markku Turunen
Nokia Research Center
markku.turunen@nokia.com

Kimmo Varpaaniemi†
Space Systems Finland
kimmo.varpaaniemi@ssf.fi

Abstract

We present an application of a modeling method for distributed systems to a case study of mobile phone camera functionality based on an existing implementation. The modeling method, Lyra, utilizes formal definitions, in this case UML2 state machines, for behavior. We observe the industrial application of the models, for instance modeling conventions and tool support needed to enable it, and the application cases of model based testing and illustration of system properties.

1. Introduction

The development of complex distributed systems benefits from using multiple kinds of analysis, for example, performance analysis and simulation. Currently, a common practice is to create a separate model for each purpose. We see two possible problems with this. First, if the system specification is informal, it may be interpreted in multiple ways, resulting in possible inconsistencies between results. Second, even if the specification is formal, the derived models do not necessarily have a well-defined relationship with the specification. To avoid these problems, we apply Lyra [11], a method for specifying distributed systems.

The specification in Lyra includes a description of the behavior of the system, in addition to definitions of interfaces between the components of the system. The behavioral description is structured in such a way as to allow straightforward linking of functional requirements to the behavioral description. The specification is created stepwise in a top-down fashion, starting from services to the user of the system.

The goal of the work was to study the use of the Lyra method for modeling of systems and their functionality for specification purposes, including executing the models in a simulator to aid system validation. Another goal was to construct the models in such a way that they could be used also for other purposes. The system used for this case study is the S60 software platform and its camera functionality. The produced models were used for a model based testing trial. In this paper we concentrate on describing the modeling of the system.

The rest of the paper is organized as follows: in Section 2 we present an overview of the Lyra method. In Section 3 we describe the modeling activity for the camera functionality and the results. Section 4 concludes the paper.

1.1. Related Research

Large industrial case studies on modeling, testing and verification by means of hierarchical state machines are reported by [1, 2, 4, 13, 15], of which [2] is the closest to our work: the process of modeling is accepted to have goals that may be poorly compatible with the goal of testing or verification. The testing approach [1] is the second closest: testing is inherently quite tolerant of the above-mentioned "other goals". The articles [4, 13, 15] consider formal verification and contain results that should encourage software engineers to take formal verification as seriously as hardware engineers have done all along.

1.2. Preceding Application Research

Lyra has been used in two case studies, modeling a 3GPP protocol system [10], and SpaceWire interconnect [9]. Both case studies focus on modeling protocols, making this case study the first to apply Lyra to modeling of software architecture. Both the 3GPP and SpaceWire case studies outline

∗ The work for this paper was done when the author was employed by Nokia Research Center.

† The work for this paper was done when the author was employed by Laboratory for Theoretical Computer Science at Helsinki University of Technology.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

some possibilities for verifying the models, a subject which is not considered in this work. The 3GPP model is used as an example case also for studying the criteria for consistency of model refinement in [12].

2. Lyra Method

Lyra is a rigorous design method developed at Nokia Research Center for development of communicating and distributed systems [11]. Modeling of systems with Lyra is service-oriented, independent of languages and built on top of formal methods. Produced models are executable, by which we mean that the models are detailed enough to run on the execution platform chosen for the purpose. For example, in the case of the S60 camera model, the execution platform is the simulator in the Tau tool.

Application of the method produces different kinds of views to the system[1]: *Functional architecture* describes in high level what a *system* can do in a platform and implementation independent manner. This description includes what are the *services* that the system provides, how the services are decomposed into functional *service components*, and what are the relations between service components within the system, and relations to services external to the system.

Platform architecture describes possible targets for service deployment by specifying what are the *system components* that a system consists of and how they communicate. The abstraction level of the system components depends on the application domain. For example, the system components can be network elements in a cellular network architecture, software and hardware in a device architecture, or software components in a software architecture. The system components can also be composed of components, and therefore are systems at a more detailed level of specification.

Logical architecture describes one possible deployment of the service components over a certain platform architecture. The abstraction level of the logical architecture is dictated by the related platform architecture.

Lyra has four iterative design phases, illustrated in Figure 1, to produce these different views: The *Service Specification* phase (*SSp*) describes the services provided by the system and the users of the services. A provided service is described by specifying how to access the service via *Provided Service Access Point* (PSAP) and what is the externally observable behavior (*PSAP Communication*) of a service via the PSAP. This phase produces the starting point of the functional architecture.

The functional architecture of the system services is defined in the *Service Decomposition* phase (*SDe*). System level services are decomposed step-wise into a set of service components. The interfaces and behavior are refined in parallel in each decomposition step while keeping the externally observable behavior consistent with the service specification. *Execution control* behavior of service components specifies how the service components are orchestrated to produce together the provided service. In addition, the use of external services via *Used Service Access Points* (USAPs) is specified (*USAP Communication*). The end result of this phase is the functional architecture of the system.

In the *Service Distribution* phase service components are deployed over system components. This may result in further decomposition of the service components. Execution control of the service is distributed with the service components, but the system wide execution control remains the same as in Service Decomposition phase. The end result of this phase is the logical architecture of the system (in some abstraction level).

In the *Service Implementation* phase a logical architecture of a system is adapted to a more concrete platform architecture. Abstraction level can still be quite high, for example, in the case when the adaptation states how logical communication between cellular network nodes is realized using communication protocols over some communication media. On the other hand, the abstraction level can be close to real implementation, for example, in the case when logical communication interfaces between software architecture components are mapped to programming languages and implementation frameworks. It is this mapping that provides an unbroken linkage from services specified in high abstraction level to an implementation in software and hardware.

The worst-case computational complexity of checking consistency of a refinement step in Lyra is "explosive", as can be concluded by looking at [8]. The current practice of Lyra ignores the complexity by assuming the following simple scheme for detection of inconsistent refinement steps: state spaces of "bounded versions" of behavioral descriptions are compared w.r.t. an appropriate behavioral equivalence or preorder. "Checking refinement consistency instead of guaranteeing it" occurs in [5, 6] and in many other papers, up to the extent that it is somehow beyond the scope of the present paper to try to find a reference where everything in the refinement would really be based on a disciplined way of transformation.

2.1. Mapping Lyra to UML2

The Lyra concepts have been mapped to UML 2.0 [11, 18] so that definitions can be presented in a standard fashion. Use of tools is necessary if one wants to construct consistent and complete models for industrial use. We used Telelogic TAU G2 [17], as it was the only tool at the time

1 A system is in this context understood to be the boundary for the entity that is to be modeled

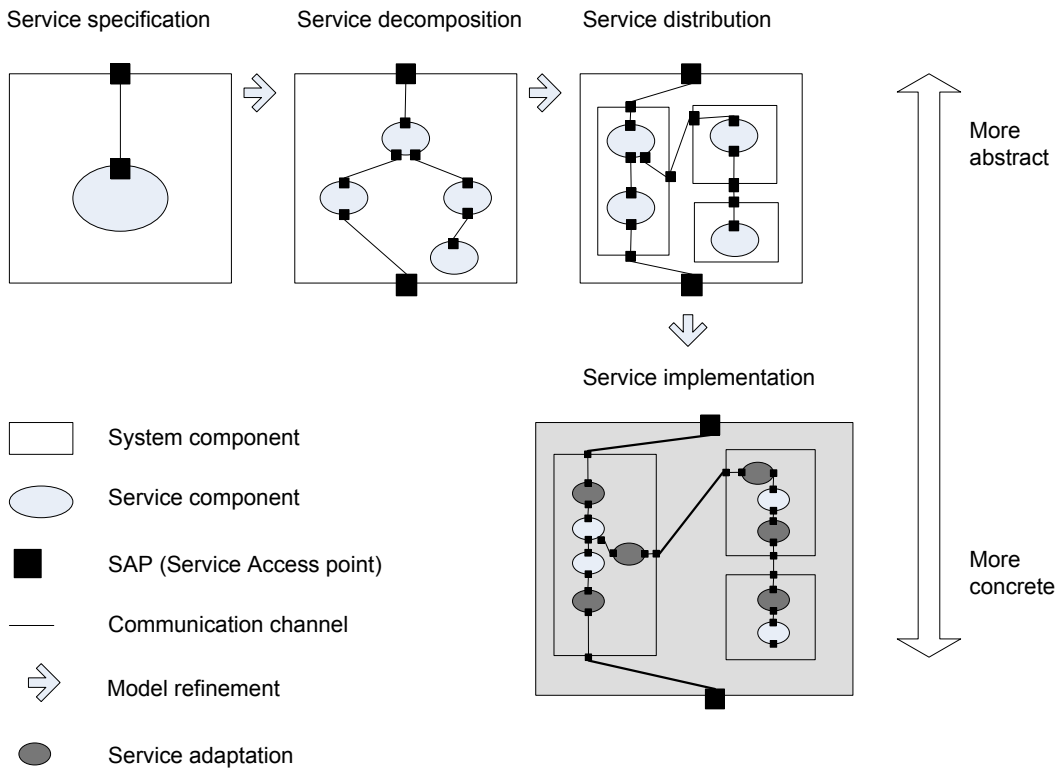


Figure 1. An Overview of Lyra phases and their relation to each other

providing all necessary capabilities needed to express Lyra models.

The Lyra concept of a service is represented as an active class [18] containing a state machine which specifies the externally observable behavior of the service. The service components are represented as state machines in either a composite state or in an active class. The system components and platform elements are represented as active classes containing services. The communication between services is performed through ports which are stereotyped according to the type of communication (e.g. PSAP or USAP communication).

The state machines used to specify the behavior of the service have a rigidly specified structure. The PSAP Communication state machine specifies the externally observable behavior on the PSAP. The composite states corresponding to the service components contain in SSp phase only a non-deterministic choice for the outcome of the service. In SDe phase, the service components are decomposed to specify the necessary steps in implementing the service that the service component provides. The refinement results in creation of additional state machines under the composite states, namely Execution Control and USAP Communication state machines. The Execution Control state machines control the execution of the new service components arising from the decomposition. The USAP Communication state machines specify the externally observable behavior on the USAP.

2.2. Uses for Lyra Models

The model elements describing interfaces and behavior of a system are suitable attachment points for requirements. Furthermore, the models allow early observation of the system behavior and also the development of related specifications such as use cases and test purposes. For instance, the advantage of traceability is seen in being able to observe functional requirements reflected in the proper system context in the model based testing exercise (see Section 3.7).

3. Modeling the S60 Camera Software

S60 is a software platform for mobile devices [14]. The Figure 2 shows one view of the S60 platform [7, 16]. The *Applications* and S60 specific *UI implementation* layers lay on top of the Generic Symbian *UI Framework* and *Application services* layers. Below are basic *OS services* together with hardware platform specific *Hardware adaptation*. These layers can be considered as Lyra *system components*, i.e. the elements of a *Platform architecture*.

In our example we reverse engineered a part of the camera system containing the still image capture functionality.

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

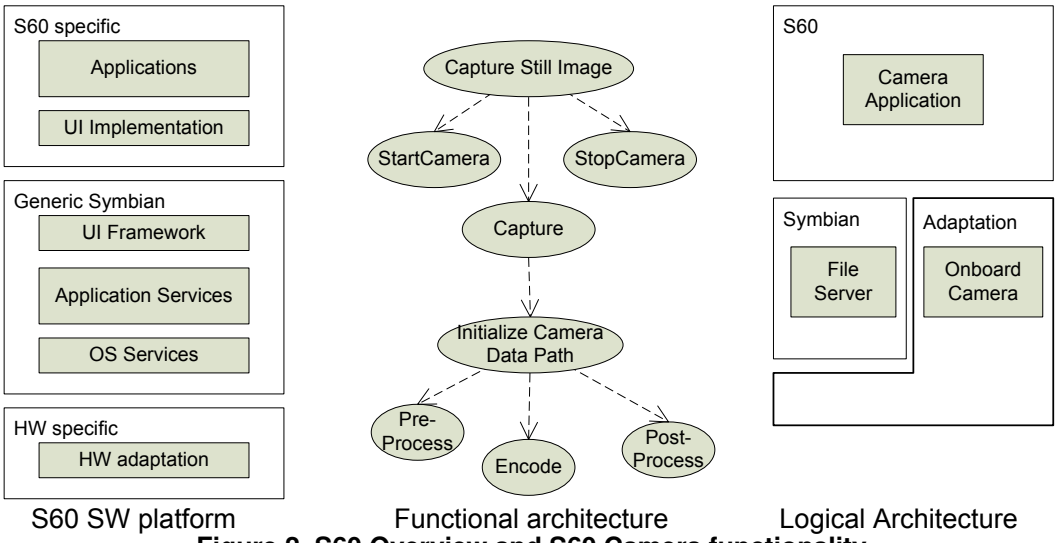


Figure 2. S60 Overview and S60 Camera functionality

Figure 2 shows the functional architecture of the modeled system (see Section 3.3) and the logical architecture resulting from mapping the functionality onto the given platform (see Sections 3.4, 3.5 and 3.6).

3.1. Model Development

If several people are to work together modeling a system, some practical issues need to be tackled in addition to the actual modeling activity. For example, the developed model needs to be partitioned in such a way as to allow parallel work and version control on the model.

The objective of cooperative use mandates some kind of partitioning of the model to identifiable objects that can, for instance be version controlled and have ownership properties. The ownership attributes are intended to hold information about authority over the modeled properties, and about responsibility toward users. The version information also allows the identification of releases of the model in development activities.

There is a clear tradeoff involved in the partitioning: a partitioning which is most suitable for version control is in our experience not easily understandable. On the other hand, a partitioning which presents the information in a model in a clear and understandable manner is not suited for version control. A solution to this problem is to generate the necessary views by scripts in the tool. Thus, the model can be partitioned in a suitable way for version control, and the scripts used to provide views illustrating the needed aspects of the system.

The system model is created by composing individual models without transformations. The approach has the limitation that the facilities of one language must be expressive enough to cover the interesting subjects of the modeled system. Model transformations are here applied only between modeling domains, for instance requirements to functions to testing. Furthermore, when using modeling tools, the composition requires that the components are modeled with such a structure that they can be nested, for example.

The validation of a composed model becomes also important considering the change over time and configuration, both on the level of a single component and the composed model. The means to validate have to extend also to the model contents, that is, some elements have to be in place to ensure consistency according to the contents and convention after changes.

3.2. Interaction between Model Components

All communication between components of a model is modeled as asynchronous message passing, as function call based communication is difficult to use with state machines. However, function or method calls in programming languages can be simulated with asynchronous communication by accepting other signals only after receiving the signal modeling the return from the function. Such simulation is best to encapsulate in separate adaptation components in Service Implementation phase.

The used signals are divided into four classes: request, confirmation, failure, indication, and response, and signal names are postfixed with req, cnf, fail, ind, and rsp, respectively. The request signals are used to ask for a service, and the response is either a confirmation (success) or failure. The indication is used when a service sends a signal to the service user without an explicit request. The user can respond to an indication with a response.

3.3. The Still Image Capture Functionality

The modeling starts with a domain model describing the services that the system offers, and the users of the services. We have restricted the behavioral modeling to the still image capture case. The model represents the actions a user can initiate. It is not a model of the actual physical user interface, nor a model of software implementation.

There is only a single type of the user, the human user. The main functionalities modeled are starting the camera, capturing an image and stopping the camera. In addition, different adjustments (e.g. zoom and brightness) are modeled, as is the standby mode which is entered if the user is idle long enough. There is one external service that the still image capture functionality uses, namely the File Service.

The domain model is formalized in a communication context diagram. Our system communicates with the user via the PSAP port and with the FileService via the FileService_USAP port.

After creating the domain model, PSAP interfaces are defined for each service offered by the system. In our case the PSAP interfaces represent the human user's actions and the software's responses to them. For example, the StartCameraReq() signal models the starting of the camera[2]. The StartCameraCnf() models the appearance of the viewfinder and camera controls. StartCameraFail() models the error message displayed if the startup fails.

During the *Service Decomposition* phase the StillImageCapture functionality is decomposed into smaller service components as shown in Figure 5. These service components show up in the PSAP Communication state machine of the service as composite states. The Figure 7 shows an example of the state machine structure, showing the Execution Control of the StartCamera service component, and the USAP Communication of the PowerOn service component.

3.4. The Camera Application System Component

In the *Service Distribution* phase the still image capture functionality is deployed over platform architecture elements. Our system component division was based on the S60 SW Platform architecture shown in Figure 2.

The *Camera Application* system component resides in the Applications layer. It contains the PSAP communication of the StillImageCapture service and use of the FileService. Hardware platform dependent part is encapsulated in the Onboard Camera system component (see Section 3.5). Camera Application uses the services provided by Onboard Camera.

2 In this case the user interface is abstract, we do not care how the action was initiated in the UI (e.g. was a shortcut key pressed or was a menu item selected)

The modeling starts with a domain model describing the services that the system offers, and external systems, see Figure 6. The domain model is formalized in a system communication context diagram. The Camera Application communicates with the user via the PSAP port, using interfaces I_to_StillImageCapture and I_from_StillImageCapture. The Camera Application also communicates with the Onboard Camera and Symbian File Server via the respective USAP ports. The interfaces on the USAP ports are defined by the systems providing the service.

3.5. Symbian Onboard Camera System Component

Symbian Onboard Camera API is an interface for controlling simple, onboard, digital camera devices. It allows capturing of still and video images. The Symbian OS provides the ECam component, defining the API, and the hardware provider must implement a software component that provides the real implementation of the API functionality.

The Symbian Onboard Camera system component contains the HW specific functionality of the Still Image Capture functionality, for example, what are the real actions needed when reserving a camera. In the *Service Implementation* phase the abstract communication between the Camera Application and Onboard Camera system components is refined toward the C++ level member function calls, *Onboard Camera API*. The domain model of Onboard Camera API in Figure 9 shows the main control interface and the call-back observer interface.

The model of the Onboard Camera API follows closely the actual C++ implementation of the API. This 1) shows how a low abstraction level model can be integrated to model of higher abstraction level 2) enables study of modeling near implementation level 3) lays groundwork for further study of generation of implementation documentation and even parts of the implementation from the model. The excerpt of the interfaces shown in Figure 8 shows how the C++ method calls were modeled using the signal naming conventions presented in Section 3.2.

3.6. Symbian File Server System Component

Symbian File Server provides operating system level access to local (ROM, RAM & removable media) and installable (e.g. remote) file systems.

The File Server component is abstracted from the quite complex interface provided by the Symbian into a simple interface that just provides basic services (like open, close, read, write, rename and delete.) This high abstraction level is selected because 1) it provides enough services for camera application model 2) it allows study of integration of a high level abstraction level model to lower level models 3)

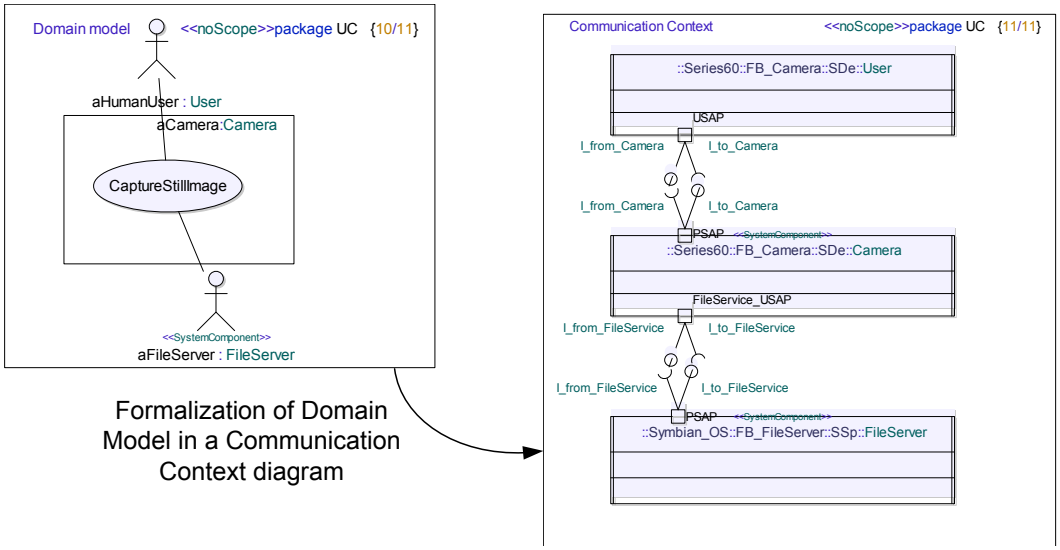


Figure 3. The relation of the Domain Model and the Communication Context diagram

```
interface I_to_Camera {
  signal StartCameraReq(part StartCameraReqPar);
  signal SetZoomReq(part SetZoomReqPar);
  signal CaptureReq(part CaptureReqPar);
  ...

}
```

```
interface I_from_Camera {
  signal StartCameraCnf(part StartCameraCnfPar);
  signal StartCameraFail(part StartCameraFailPar);
  signal SetZoomCnf(part SetZoomCnfPar);
  signal SetZoomFail(part SetZoomFailPar);
  signal CaptureCnf(part CaptureCnfPar);
  signal CaptureFail(part CaptureFailPar);
  ...
}
```

Figure 4. Part of the interface definition for Camera Application

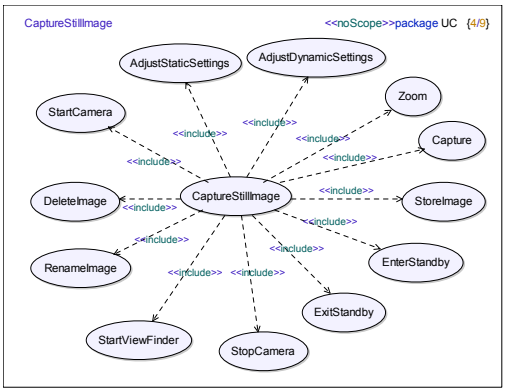


Figure 5. The Functional Decomposition of the still image capture service

creation of more detailed model would had been hard without detailed information of internals of Symbian File Server. The domain model of File Server is shown in Figure 9.

3.7. Model Based Testing

In order to evaluate the applicability of Lyra models for testing purposes, the S60 model was used in a model based testing case study. The model can be used for testing by refining them by adding information about correct parameter

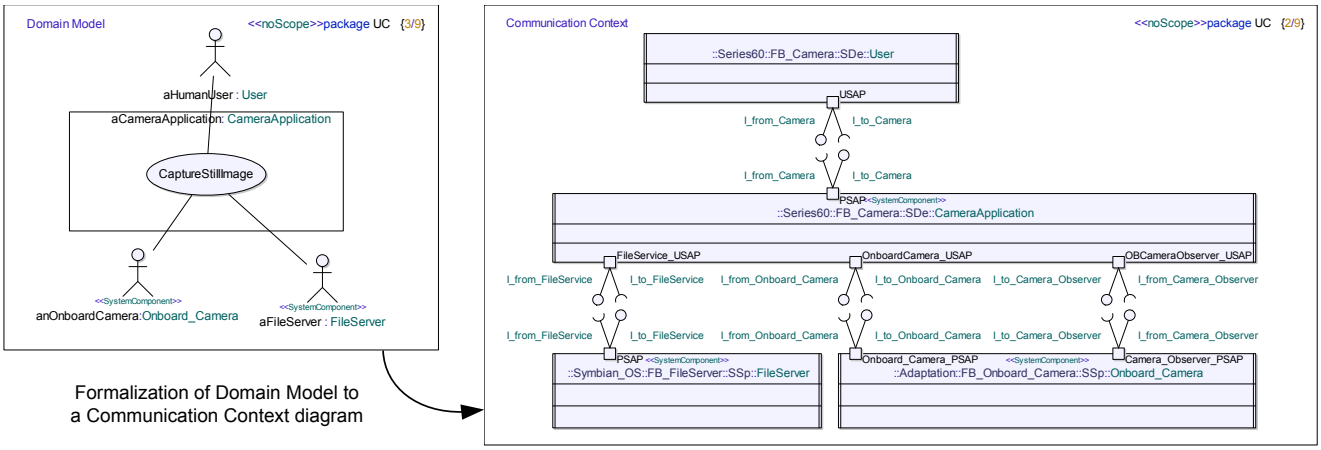


Figure 6. The Domain Model and the Communication Context diagram for the Camera Application system component

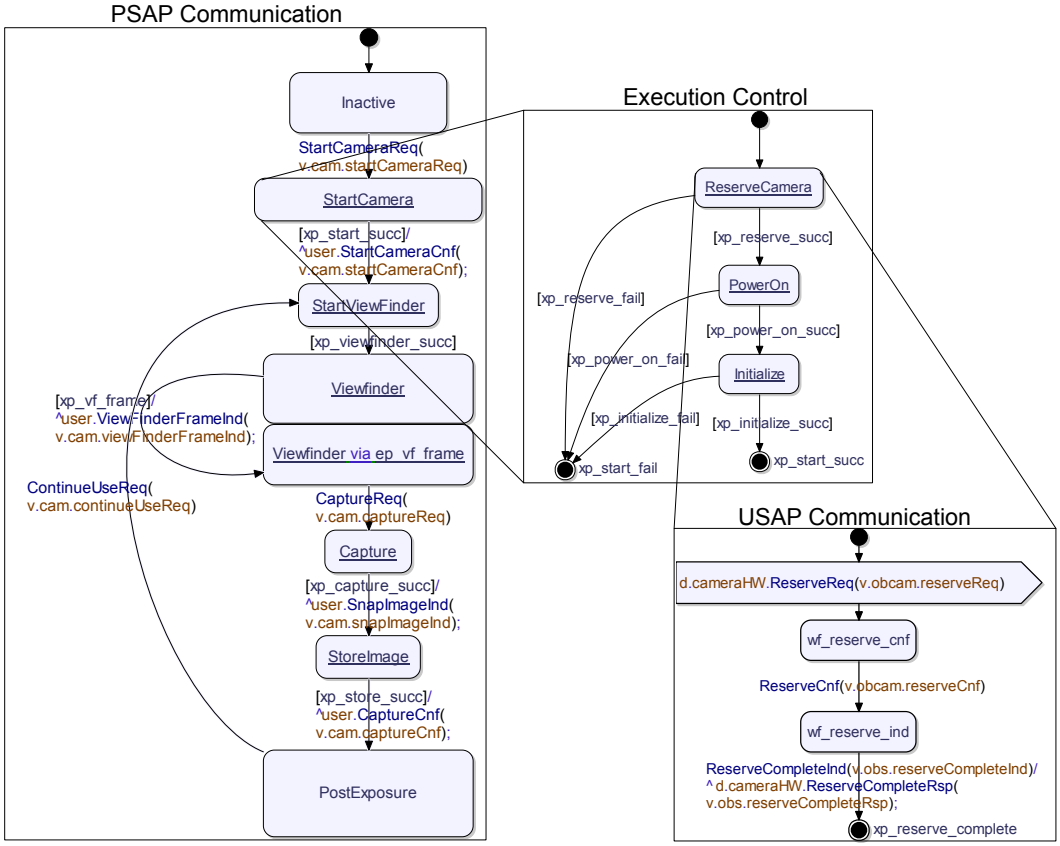


Figure 7. An example of a Lyra state machine: Camera Application. The diagram only shows a part of the complete state machine. The states with an underlined name are composite states. The triggers [xp_...] refer to exit from a composite state by a named exit point.

values. In addition, a tool infrastructure that can use system models (as opposed to test models) for testing is needed. Our model based testing trial used Conformiq Qtronic [3] pre-release tool as the testing tool. The needed workload to

```
interface I_to_Onboard_Camera {
 signal NewLReq(part NewLReqPar);
 signal CameraInfoReq(part CameraInfoReqPar);
 signal ReserveReq(part ReserveReqPar);
 signal PowerOnReq(part PowerOnReqPar);
 ...
}
```

```
interface I_from_Onboard_Camera {
 signal NewLCnf(part NewLCnfPar);
 signal CameraInfoCnf(part CameraInfoCnfPar);
 ...

}
```

```
interface I_to_Camera_Observer {
 signal FrameBufferReadyInd();
 signal PowerOnCompleteInd();
 signal ImageReadyInd();
 signal ReserveCompleteInd();
}
```

```
interface I_from_Camera_Observer {
 signal FrameBufferReadyRsp();
 signal ImageReadyRsp();
 signal PowerOnCompleteRsp();
 signal ReserveCompleteRsp();
}
```

Figure 8. Partial interface definitions for the Onboard Camera component. The parameter names have been left out from the observer interfaces.

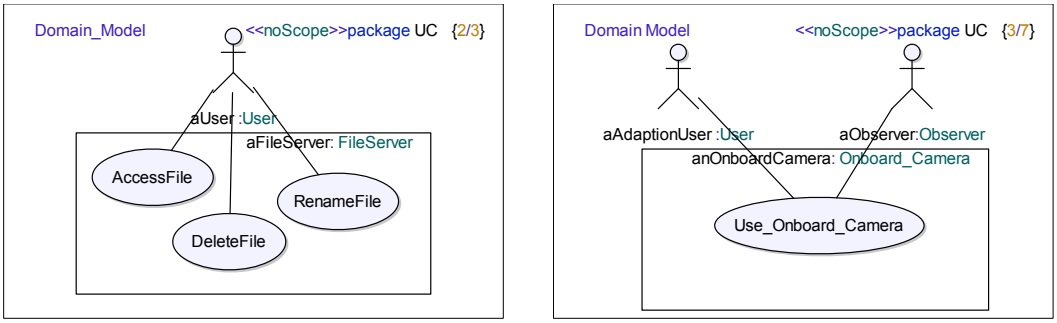


Figure 9. The domain models of File Server and Onboard Camera components

refine the models was manageable, and could probably be automated to a large extent.

3.8. Experiences in Modeling

When the Lyra method is used to reverse-engineer an existing system (not designed with Lyra), the mismatch of architecture abstraction levels, especially on behavior description, causes extra complexity in model. One shall identify what complexity comes from the modeled functionality itself and what comes from the selected implementation mechanism and then model those separately. Thus, the majority of problems were due to insufficient separation of different abstraction level concerns in the model.

We did not produce abstract service specification level description for the Onboard Camera, but started instead the description of the functionality by reverse engineering the existing Onboard Camera API. As a result the USAP communication level of the service components in the Camera Application system component became more complex than needed. Instead we should have specified all the interfaces between service components in an abstract level and then only during the Service Implementation phase deal with C++ level details. Furthermore, the details should have been separated from the core functionality by encapsulating them in their own service components.

Solving modeling problems in the applied language and tool constituted a significant part of the work. This indicates the importance of evaluating any prospective tool with the actual model contents and conventions.

Telelogic TAU G2 model execution environment has its own built in scheduler. While this scheduler can be replaced, doing that is not always practical. Trying out the model and tests with different schedulers could ensure that the model has no weaknesses because of assumptions of the scheduling of processes.

The tools should be tailored for use with a method in order to increase productivity. The tailoring includes e.g. automation of repeated modeling tasks and good integration with other used tools.

The guidelines for modeling, concerning e.g. the structure of the model, were developed during the case study. Even though model validation was not attempted during the case study, we identified a need to automatically validate the models against the method rules and conventions. Automatic update of models to reflect such changes in a reposi-

tory is necessary.

4. Conclusions

We modeled the S60 mobile phone software with the Lyra method. The resulting model was used in a model based testing trial, demonstrating that Lyra models have also other uses besides system specification. Using the method with UML2 led to the conclusions that the method can be used to produce models with variable level of detail, and that the concrete use cases should be applied in selection of the language and tools. So far the results are positive with applying the method, but we cannot conclusively state that the modeling language constructs fit without problems to all domains. Further studies can show this.

The method, as devised, is especially suited to modeling starting from the problem statement of the system. However, reverse-engineering existing systems can be beneficial if done at a high enough abstraction level. Then, the models constructed in Service Specification and Service Decomposition phases describe the essential functionality. The functionality is then mapped to a concrete implementation platform in Service Implementation phase.

Applicability of the models can be further studied, for instance, for analysis to verify selected system properties, and to validation of models by model checking mechanisms.

In practice, the Service Specification phase models of the components were felt to be most useful. They specify the protocol for service usage unambiguously with a PSAP Communication state machine, serving as a communication aid for the creators and users of a component. Furthermore, as the Service Specification phase models only describe the externally observable behavior of a service, the models are kept small and easily understandable.

5. Acknowledgements

The authors would like to thank Stephan Schulz for providing helpful comments on the appropriate abstraction level of the components, and Sami Heinonen for modeling the File Server component.

References

[1] Chevalley, P., and Thévenod-Fosse, P.: "An Empirical Evaluation of Statistical Testing Designed from UML State Diagrams: the Flight Guidance System Case Study." In: *Proceedings of the 12th International Symposium on Software Reliability Engineering*. IEEE Computer Society Press, 2001, pp. 254–263.

[2] Cigali, S., Leblanc, P., Malaponti, S., Mandrioli, D., Mazzuchelli, M., Morzenti, A., and Spoletini, P.: "An Experiment in Applying UML2.0 to the Development of an Industrial Critical Application." In: *Proceedings of the 2nd Workshop on Critical Systems Development with UML*. Technical Report TUM-I0317, Technical University of Munich, Germany, 2003, pp. 19–33.

[3] www.conformiq.com.

[4] Damm, W., and Klose, J.: "Verification of a Radio-Based Signaling System Using the STATEMATE Verification Environment." *Formal Methods in System Design*, Vol. 19, No. 2, 2001, pp. 121–141.

[5] Davies, J., and Crichton, C.: "Concurrency and Refinement in the Unified Modeling Language." *Formal Aspects of Computing*, Vol. 15, No. 2–3, 2003, pp. 118–145.

[6] Engels, G., Heckel, R., Küster, J.M., and Groenewegen, L.: "Consistency-Preserving Model Evolution through Transformations." In: *UML 2002*. LNCS, Vol. 2460, Springer-Verlag, 2002, pp. 212–227.

[7] www.forum.nokia.com > S60 Platform > Tools/SDKs > S60 Platform SDK

[8] Harel, D., and Kupferman, O.: "On Object Systems and Behavioral Inheritance." *IEEE Transactions on Software Engineering*, Vol. 28, No. 9, 2002, pp. 889–903.

[9] Honkola, J., Leppänen, S., and Tynjälä, T.: "Modeling the SpaceWire Architecture with Lyra." In:*Fifth International Conference on Application of Concurrency to System Design, 7–9 June 2005, St. Malo, France*. IEEE Computer Society, 2005, pp. 15–24.

[10] Leppänen, S., Turunen M., and Oliver I.: "Application Driven Methodology for Development of Communicating Systems." In: *FDL'04 ECSI Forum on Specification and Design Languages, September 14–17, 2004, Lille. France.*

[11] Leppänen, S.: "The Lyra Design Method". Report 40, Institute of Software Systems, Tampere University of Technology, 2005.

[12] Leppänen, S., Ilic, D., Malik, Q., Systä, T. and Troubitsyna, E.: "Specifying UML Profile for Distributed Communicating Systems and Communication Protocols." In: *Workshop on Consistency in Model Driven Engineering - C@MODE05, Nov 2005.*

[13] Ober, I., Graf, S., and Lesens, D.: "Modeling and Validation of a Software Architecture for the Ariane-5 Launcher." In: *FMOODS 2006*. LNCS, Vol. 4037, Springer-Verlag, 2006, pp. 48–62.

[14] www.s60.com

[15] Sharygina, N., Browne, J., Xie, F., Kurshan, R., and Levin, V.: "Lessons Learned from Model Checking a NASA Robot Controller." *Formal Methods in System Design*, Vol. 25, No. 2–3, 2004, pp. 241–270.

[16] www.symbian.com > Symbian OS

[17] Telelogic TAU G2 documentation. URL:http://www.telelogic.com/.

[18] OMG. UML 2.0 Superstructure Specification, 2005.

Application of Bayesian Networks to Architectural Optimisation

Artem Parakhine[1,2], Tim O'Neill[2] and John Leaney[2]
1. Faculty of Information Technology
2. Institute for Information and Communication Technologies
University of Technology, Sydney
aparakhi@it.uts.edu.au
tim.oneill@uts.edu.au
John.Leaney@uts.edu.au

Abstract

The field of optimisation covers a great multitude of principles, methods and frameworks aimed at maximisation of an objective under constraints. However, the classical optimisation can not be easily applied in the context of computer-based systems architecture as there is not enough knowledge concerning the dependencies between non-functional qualities of the system. Out approach is based on the simulation optimisation methodology where the system simulation is first created to assess the current state of the design with respect to the objectives. The results of the simulation are used to construct a Bayesian Belief Network which effectively becomes a base for an objective function and serves as the main source of the decision support pertaining to the guidance of the optimisation process. The potential effects of each proposed change or combination of changes is then examined by updating and re-evaluating the system simulation.

1. Introduction

The ongoing increase in the complexity of common computer-based systems is evident from such trends as rapid growth of the application server market and fast acceptance of aspect-oriented programming. These new technologies and methods are aimed at dealing with unexpected behavior which emerges when large sets of functional elements are integrated to form a single system. These problems may be caused by the structural properties of the overall system or by the faulty assumptions made during creation of individual elements.

As such it has become important to make systems that not only conform to their functional specification, but also possess an array of non-functional properties which may vary depending on their application domain. Additional complications arise in cases when the desired non-functional properties are in conflict with each other. Consequently, it is imperative to ensure that the process of system design possesses faculties to manage such conflicts and prevent any single non-functional quality from taking overwhelming precedence over others.

In previous work [16] we proposed a framework for system design, which is based on *heuristic* approach to design and optimisation. By choosing to build the process of optimisation around iterative application of *heuristic*-encoded design decisions we propose that it will be possible to find a solution which represents an optimal compromise on competing system qualities. An brief overview of the framework is presented in Figure 1 below.

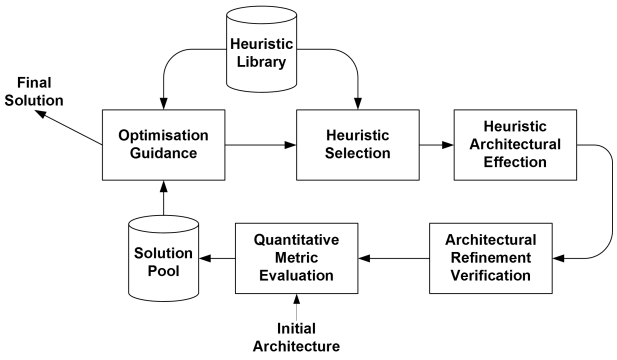


Figure 1. Overview of heuristic-based optimisation framework

The process based on the proposed framework first evaluates the initial architecture to determine how far it is from the specified goals. Following that the "Optimisation Guidance" component has to determine the constraints to be

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

used by the "Heuristic Selection" module to find a *heuristic* most likely to improve current architecture. At the next step of the process the change described in the chosen set of *heuristics* is affected and the outcome is verified to functionally correspond to the original system by the "Architectural Refinement Verification" [7]. Finally, before the loop-back the resultant architectures are evaluated with respect to the original goals and made available to the guidance component via the solution pool.

The idea is to make the proposed process extendible by allowing incorporation of various, possibly domain specific libraries of *heuristics* as well as addition of evaluators to the "Quantitative Metric Evaluation" module. Such approach would allow various modules of the framework to advance with high degree of independence. Additionally, the proposed framework allows for architectural optimisation of the system design at an early stage and would positively affect the cost and duration of the overall development process.

Under the proposed configuration for the optimisation framework the responsibility of dealing with the multitude of available heuristics and metrics is assigned to "Optimisation Guidance" and "Heuristic Selection" modules. Within these modules previously performed steps and their effects on the architecture must be analysed and used to determine which parts of the architecture must be addressed next. Guidance components must also determine the best possible alternative architecture in the current solution pool and which combination of *heuristics*, when applied, would produce an outcome closes to the desired solution.

The approach used to work out the aforementioned elements of the optimisation framework based around a modeling methodology employing Bayesian Networks as the main decision aid is the focus of this paper. The content is structured as follows: Section 2 provides a survey and overview of related concepts. Section 3 contains details of our proposed methodology and Sections 4 and 5 provides a discussion of conclusions and future work.

2. Architectural Optimisation Framework

The problem of providing meaningful guidance to a process aimed at optimising an architecture of a complex computer-based system can be represented as one of a multidisciplinary design optimisation. In recent times attempts have been made to describe the problem of architectural change as one aiming to fulfill multiple objectives handled by distinct disciplines.

Proposed approaches covered both bottom-up [1] and top-down [12] techniques. In the case of the latter the multiple objective aspect has been addressed from the point of view of possible trade-offs between cost and benefit of the proposed change and possible effect it has on other, potentially conflicting objectives. In our methodology we propose to follow the principles of multidisciplinary approach described below.

2.1. Multidisciplinary Design Optimization

It is an accepted fact that the problems faced by professionals working on designs of such systems as road vehicles or aircraft can be best represented as complex interaction between system parts and physical phenomena best described by various disciplines [14], [13]. Hence, first proposed by NASA [5], the field of Multidisciplinary Design Optimization (MDO) has evolved as a new discipline that provides a body of methods, techniques and some CAD tools to assist engineers in devising a near-optimum system design in multiobjective environment.

The existing MDO methods are based around the idea of decomposition of the design task into two distinct levels: **subtasks** which can be performed independently in each of the modules and **coordination** or system-level task.

Original motivation for such decomposition was to satisfy the obvious need to distribute work over multiple groups of individuals, and thus shorten the overall project duration. Therefore the important effect of the decomposition is granting autonomy to the groups of engineers responsible for each particular subtask as well as allowing for the concurrent execution of the subtasks.

Using this philosophy as a guide we tried to determine the best way of classifying various architectural qualities into independent groups within the context of the optimisation process.

Multidisciplinary design optimisation aims to improve the sought after qualities within the system and supports management of trade-offs between the design objectives described by diverse disciplines. To fulfill the requirements of such optimisation in the field of systems architecture we propose a framework based on optimising a design through simulation using heuristic application supported by a Bayesian belief network.

The proposed framework is aimed at operating with two major artifacts: the design of the system in question and a collection of heuristics. The elements of this collection are expected to present options for structural or property changes which can be done to the system design with the intention of positively affecting one or more of its non-functional qualities. The overall aim of this framework it to propose a list of changes which will have a high likelihood of improving the desired non-functional qualities of the system. Therefore, we are seeking a way of classifying the heuristic-encoded system changes with respect to their suitability for optimisation goals.

Traditionally, classification is based around a set of rules, which represent a set of criteria applied to the properties of

the elements in the process of classification. However, in simulation-based optimization, it is the process itself and its optimisation goals that provide the means to determine element suitability. These rules possess a higher degree of complexity and have an iterative character.

In the proposed framework the process of classifying a new heuristic element, not yet examined, would take the following form. If the element is already a part of a collection of architectural modification to be treated as a single modification, then it has to be decomposed and applied within the context of the simulation process to evaluate it, and to see where it falls in relation to previously classified heuristics. Consequently, if a particular heuristic has been applied to the design in question then the simulation has to reflect the change to the design and the optimisation process will be re-run in order to incorporate possible new conditions and constraints introduced with the change.

The notion of optimisation encapsulates issues of identifying best values for a set of decision variables. However, available optimisation methods have difficulty coping with complexities and uncertainties posed by many real world problems, which can be successfully simulated [9]. The need for optimisation of simulation models arises when the designer wants to find a set of model specifications, such as individual or structural available, that leads to optimal performance.

In a situation when a complex system change is considered, the range and number of possible parameter combinations is often too large for designers to simulate. Therefore, they need a method to intelligently guide the search for an optimal solution. However, without simulation, many real world problems are too complex to be modeled by mathematical functions that are at the core of pure optimisation methods. Hence, in order to guide the simulation and enrich the optimisation with an account of the complexities and dynamics of the system, one must resort to a hybrid approach.

The merging of optimisation and simulation technologies has seen growth in recent years [9]. Until relatively recently, however, the simulation community was reluctant to use optimisation tools. However, recent developments in the field of metaheuristics provide a possibility for future integration. The concept of 'metaheuristics' refers to a domain of optimisation research which aims to enhance the traditional mathematical approach with methods based on analogs to physical or evolutionary processes [17] as well as artificial intelligence research. The introduction of metaheuristics have enabled creation of optimisation engines capable of employing series of complex evaluations to determine the optimal values for the decision variables [6].

In the proposed framework the role of the bridging element used to represent the knowledge about dependencies, which may be relevant to the task of system optimisation, will be fulfilled by a Bayesian Belief Network (BBN) which are described in greater detail in Section 2.2.

In recent years, there were several attempts undertaken by researchers to represent the knowledge about the relationships between system qualities using BBNs. Gurp and Bosch [22] proposed a variant of such network. SAABNet (Software Architecture Assessment Belief Network) is aimed at helping developers perform qualitative assessment during the architecture design process. Their proposed network is comprised of two types of nodes: one which represents various system characteristics and their interdependencies, eg. component granularity which depends on implementation language and architectural style. Another type of node present in SAABNet serves as a representation for external quality factors such as reliability and useability. The interdependencies between various nodes described by Gurp and Bosch are based on a modified version of McCall's Factor-Criteria-Metric model.

The operation of SAABNet, as described by Gurp and Bosch, is aimed at providing a set of values which would assist the designer in selecting system properties based on quality goals. The advisory set generated by the system contains the values for architectural characteristics such as interface and component granularity which are highly likely to contribute towards rendering a system which possesses the desired qualities of configurability and understandability.

Additionally, SAABNet can provide the designer with information about variables that will need special attention during the development process as they may carry a lot of dependency relationships [21]. Finally, by examining the causality links between various nodes in SAABNet, the designer can effectively identify possible conflicts and tradeoffs which exist in the architecture. All of these applications are necessary for a successful optimisation process.

Another way of looking at the application of BBNs to describe the architectural qualities is proposed by Trendowicz [19]. This new approach differs from SAABNet in the general perspective it adopts in the task of describing criteria-quality causalities in the system. The main feature of the methodology proposed by Gurp and Bosch is that it aims to construct a network which contains all considered system qualities. However, Trendowicz chooses to focus on individual qualities in greater detail.

Both of the aforementioned approaches take advantage of properties of BBNs that make them applicable for quality modeling. The ability to describe complex dependency in a simple graph form with associated probabilities make BBNs configurable to suit the need of an individual system as well as extendible to cover the a domain of system application. Additionally, application of Bayesian probability theory to describe qualitative estimations of system properties intro-

duces the necessary mathematical rigor to the process and creates an opportunity for optimisation.

However, a serious hindrance to incorporation of BBNs into an optimisation process is caused by the assignment of probability values associated with quality dependencies. Existing methodologies propose to define these values through knowledge elicitation and static data examination. However, this creates a possible problem as the likelihood values are skewed by assumptions compounded along the dependency lines. Furthermore, such approaches do not provide adequate facilities to track how the probability values may vary as a result of optimisation-driven change.

2.2. Bayesian Networks

A Bayesian Belief Network (BBN) is a directed acyclic graph that represents a probability distribution [11]. An example of a simple BBN can be found in figure (insert ref here).

The elements present on the graph carry the following meanings:

Nodes represent random variables; quantitative probability information is specified in the form of conditional probability tables describing the probability of each possible state of the node given each possible combination of states of its parents.

Arcs represent probabilistic correlation between the variables; the presence of an edge (or lack thereof) between two nodes indicate probabilistic relationship.

The main useful characteristic of BBNs is that they provide a method for decomposing a general system-wide probability distribution into a set of local conditional distributions. Once constructed, the topology of the network can be further analysed to determine partial probability distributions pertaining to the chain of causality which may be of particular interest in the current context.

This particular feature of BBNs has some important consequences:

- In large systems where interaction between observed variables are sparse, BBNs dramatically reduce the amount of work required to specify joint probability distribution.
- Localised distribution representation is open to creation of efficient inference algorithms.

However, it should be noted that for large systems involving multiple observed variables which in turn affect multiple qualities, the effort involved in building a static BBN is impracticable from an engineering standpoint. Although large domain models can often be decomposed into a collection of independent smaller networks, it is desirable, for systems that need to address a broad range of problems, to be able to establish the causality links between model nodes dynamically.

The resultant BBN representation of the joint distribution does not possess any generic modularity and therefore is not easily open to reuse in other domains of system application. However, a major advantage of BBN is that it allows mixing discrete and continuous variables to generate a hybrid inference. This is especially useful since software engineers are used to think in the discreet manner even about variables that may be continuous in nature.

The optimisation framework we propose adheres closely to the principles of Multidisciplinary Design Optimisation. Within it, the function of the overall **coordination** task is performed using information accumulated as part of the *Principal BBN* described in Section 2.2.1. Furthermore, we propose to achieve decomposition into **subtasks** using *Auxiliary BBNs*, which provide information on issues associated with individual qualities or a luster of closely related qualities described by a single discipline. The BBNs will be used by the framework to identify the probability of obtaining a certain value for a system quality given the evidence collected via simulation or entered by the user. Consequently, the guidance component of the framework will attempt to direct the process of optimisation with the intent to maximise the probability of a better value for the target system quality.

2.2.1. Principal BBN First one is represented by the *principal* Bayesian Belief Network (BBN) which encapsulates the main inference relationships which are present in the architecture with respect to the goals of current optimisation. It is built with the same underlying principles used by Gurp. (Insert reference to an example figure). In other words, it is comprised of three distinct groups of nodes:

simple characteristics These nodes describe the observed characteristics of the system known to be relevant to the goals of the system optimisation (insert figure reference);

complex characteristics Members of this group represent the knowledge of how the simple characteristics combine into more complex ones which in turn produce cumulative effect onto the system qualities (insert figure reference)

system qualities Final layer of nodes which is created to compose the inference structure about the system qualities pursued by optimisation goals (insert figure reference).

The *principal* network is created to represent the core inference structure required to control the direction of the optimisation process. The measures of the expected effects of heuristic application will be taken from it and used to evaluate the comparative suitability of possible optimisation

scenarios. It is, therefore, a Bayesian Network representation of the objective function for the overall process.

2.2.2. Auxiliary BBNs The second entity in the interaction depicted in Figure (reference the figure) is a collection of *auxiliary* BBNs. This entity is used to catalogue the existing knowledge about causality relationships between system elements and qualities which lie outside the scope of the current optimisation goals.

A typical *auxiliary* network follows the same principles used by Trendowicz [19]. Hence, in functional terms, networks of this type address the inference knowledge pertaining a specific quality or a tightly coupled group of qualities affected by a restricted set of simple and/or complex system characteristics.

The *auxiliary* BBNs are necessary primarily to ensure that constraints of optimisation are properly addressed within the process. Depending on the conditions imposed by the problem these constraints may be considered to be **hard** or **soft**. The former type refers to ensuring that application of heuristics does not dramatically reduce a known quality of the system, whereas the latter is there to show what effects a certain course of action will have on qualities not addressed in goals.

Since each node in a BBN represents a variable which may assume a value according to the associated conditional probability distribution, it is not possible to express general relationships within the system without enumerating all the potential states every node may assume in advance. However, in the process of gathering this information it is necessary to employ methodologies, such as Markov Blanket (MB) [2], which help to remove attributes that are unnecessary.

2.3. Markov Blankets

The definition of a Markov Blanket for a node x is the set *MB(x)* which includes a set P_x of direct parent nodes of x, a set C_x of direct children nodes of x and a set S_x of spouse nodes of x [20]. A spouse node x_{si} is defined as one for which there is a child node $c_i \in C_x$.

Every node in the network is conditionally independent of x when conditioned on the set *MB(x)*. Formally, for distinct nodes x and y:

$$Pr(x|\, MB(x) \cap y) = Pr(x|\, MB(x)) \qquad (1)$$

Markov blanket criterion only removes attributes that are really unnecessary: attributes that are irrelevant to the target node, and attributes that are redundant given other attributes.

Finally, several algorithms exist [20] which use the concept of Markov blanket to guide Bayesian Network construction. The common first step of such algorithms involves identification of *MB(x)* for all x, from that point the discovered information is used to guide the construction of the Bayesian Network in the current context. In the example presented in Section 3 an instance of a Markov blanket is used to isolate nodes of interest, however no method for automatic discovery is employed as it lies outside the scope of this paper.

2.4. Multiparadigm Simulation

In some cases it is impossible to actually work out a definitive measure for the non-functional quality of the system. But use of Bayesian networks makes it possible for us to introduce elements of correlation and likelihood which can be either explained by some sort of functional relationship or by accidental observation.

In order to initiate the discovery of causal relationships which may exist in the system to be optimised we need to produce a simulation model which can be mined for characteristic information. There are three major paradigm which currently exist in the field of simulation modeling [4]:

1. System Dynamics (SD) - deals at the highest abstraction level with aggregate values, global feedbacks and trends.
2. Discrete Event (DE) - deals with resource allocation, schedules, levels and capacities at medium to high abstraction levels.
3. Agent Based (AB) - characterised by presence of active objects with individual behaviour rules within configurable environment.

In SD the basis for representation is formed by three elements: description of *stocks* of items, *structure* to describe flow between the stocks and *policies* that determine the properties of the flows such as value and volume. From SD perspective system behaviour has to be described as a number of balancing or reinforcing feedback loops [8].

The DS approach is based around *entities* which are passive objects that represent tasks, messages, etc. They travel through the blocks of the flowchart where they are delayed in queues, processed, split or combined, seize and release resources.

There have been a great variety of advancement in the field of Agent Based modeling in disciplines like artificial intelligence, complexity science and game theory [4]. What sets the agent based simulation apart from other types described above is that it is essentially *decentralised*. In contrast to SD and DS approaches, AB paradigm does not require the precise knowledge of the system operation. Instead, AB adopts a bottom-up mechanism of achieving emergence of global system properties through definition of characteristics comprising the system, which can be successfully employed in the process of discovery described in Section 2.3. Such characteristics of the AB paradigm

create a possibility of using it to cover all other simulation paradigms.

Additionally, Agent Based modeling can be used across all abstraction levels. Agents may model objects of very diverse nature and scale [3]: at the physical level agents may be pedestrians or cars or robots, at the middle level customers, at the highest level competing companies.

However, once the simulation is complete in order to proceed with optimisation process we have to work out a mechanism for affecting change.

2.5. Heuristics

The classical optimisation approach of finding a maximum or minimum value of a function of several variables subject to a set of constraints may not be easily applied in the context of architectural optimisation. The problem arises from the fact that the non-functional properties of a complex software system stem not only from the parameters of its comprising elements but also from its structure. In earlier work [16] we have endeavored to postulate that a heuristic based approach to effecting change as part of system optimisation process may be more suitable for our needs.

The prescriptive nature of heuristics has been described by Rechtin [18]. In his work on systems architecture he identified two categories of heuristics, *descriptive* and *prescriptive*, which exist at all levels of systems engineering and in all domains. Furthermore, some work has been done by Grunske [10] who proposed a methodology based on graph transformation rules aimed to transforming a system topology with the purpose of improving its safety characteristics.

In our view a single heuristic represents a suggested solution to resolve a specific deficiency. However, a heuristic doesn't necessarily carry a suggested solution. Nevertheless it can still be used to identify the deficiency and as such plot out further direction for optimisation by reinforcing an adoption of another heuristic which actually addresses (if only partially) the identified system shortcomings.

Additional work has been done by Maxwell [15] to enunciate how knowledge aggregating properties of heuristics can be formalised and incorporated into an architectural optimisation framework. Formalisation is required to ensure that the functional properties of the system can be verified through refinement after the chosen heuristic has been applied. To that end the graph representation proposed by Maxwell [15] carry the advantage of having formal rigor.

From graph theory perspective an option proposed by a heuristic is expressed as a transformation rule that can be applied to a location within the architecture given that the *a priori* conditions and negative application conditions hold.

3. Example Application

This section attempts to illustrate how an agent-based simulation can be used to determine likelihood values along the causality arcs present in the *principal* and *auxiliary* BBNs described in Sections 2.2.1 and 2.2.2.

First, we adopted SAABNet described by Gurp [22] as a basis for the *principal* network in our framework. For this example we decided to focus on the "Useability" quality and its parent nodes. This was done by considering the Markov blanket of the "Reliability" node which is believed to have a strong influence on "Useability" in the example system. The resultant BBN is depicted in Figure 2. The network and its environment has been constructed using a GeNIe tool (http://genie.sis.pitt.edu/) which provides a simple and clean interface for creation and management of Bayesian networks.

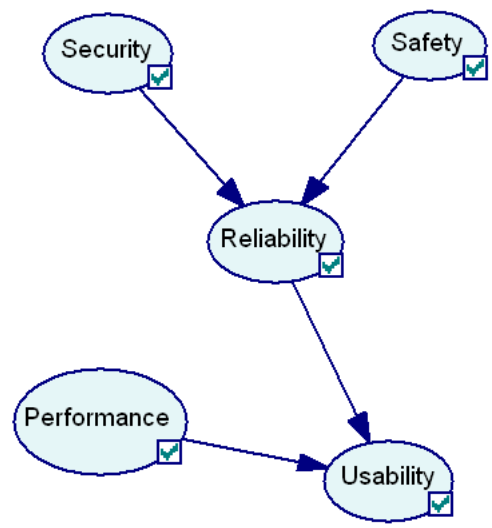


Figure 2. Markov Blanket network for Reliability

For the purposes of this example we have decided to disregard "Configurability" and concentrate on "Reliability" and "Performance", which are described in Table 1.

quality	P(good)	P(bad)
reliability	0.99	0.01
performance	0.86	0.14

Table 1. Likelihood values for Reliability and Performance

At this point, in order to determine how these defined values affect the "Useability" quality we had to employ the Agent-Based paradigm approach described in Section 2.4. In order to achieve that we had to create an agent which could apply them in a meaningful way in the context of the system. The state transitions describing agent behavior are shown in Figure 3, which has been generated using AnyLogic tool (http://www.xjtek.com/).

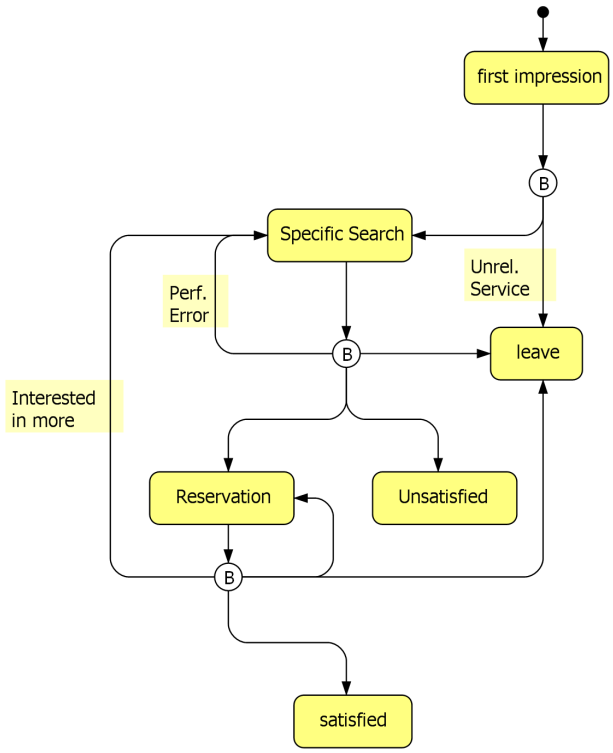


Figure 3. Behavior Description for Useability Agent

The agent represents a typical user of the reservation service developed and maintained by an Australian company. This is a simplified version which covers only the most basic use-case scenario for the system.

Several things should be noted about the described state transitions. Firstly, transitions leading to the "Leave" state on the diagram represent instances of agents discontinuing the use of service due to poor reliability. The execution of these transitions is controlled by a distinct function which combines the probability value associated with "Reliability" and the current state of the agent. Likewise, arcs leading from branch elements back to originating node represent retries due to poor performance and are also controlled by individual probability-based functions.

After several runs of the simulation have been completed, the following scalar results have been obtained for a population of 1000 agents:

- n repeat requests due to timeouts = 222
- n successful requests = 1332
- **useability at good performance = 0.84**
- n failures due to unresponsiveness = 41
- n visitors = 1000
- **useability at good reliability = 0.959**

Using the values listed above we were able to come up with the combined likelihood values which describe the effect various states of "Reliability" and "Performance" have on "Useability" (Table 2).

reliability	good		bad	
performance	good	bad	good	bad
useable	0.80556	0.15344	0.03444	0.00659
not	0.19444	0.84656	0.96556	0.99341

Table 2. Likelihood values for Useability

With the information obtained as part of this example we hope to show that it is possible to incrementally populate the *principal* and *auxiliary* BBNs using a simulation. At this point we only used an agent-based simulation approach, however, given further metrics and information pertaining to system structure and company's project activities we can develop a reacher simulation incorporating principles of system dynamics and discrete events simulations. Once the BBNs have been provided with sufficient sampling information they can be used to make optimisation decisions given a collection of heuristic encoded changes.

4. Conclusion

Recent research is promising to solve the problems associated with existing approach to architectural quality modeling. The properties of Bayesian Belief Networks position them as a useful aid in handling knowledge organisation, systemic reasoning and quality prediction opening a possibility of application as a basis for optimisation. However, the major drawback of BBNs is found in the amount and characteristics of the construction effort required before they can be used.

In our work, we propose a way of dealing with this drawback by incorporating multiparadigm simulation approach to capture the assumptions, properties and structural elements which exist in the context of systems architecture.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

By doing so we propose to develop a mechanism which can be used to trial possible changes before they can be applied and evaluate their potential effects given the likelihood and dependency information provided by BBNs.

5. Future Work

Following the discussion of issues and the accompanying example presented in this paper, we believe the next step is to develop a broader range of examples to cover systems from multiple domains. In time this would allow us to study possible patterns of metric-criteria-quality dependencies which may consistently emerge in similar systems. Another avenue of research is to attempt application of automatic Markov blanket discovery algorithms to known quality assessment BBNs. Additionally, work will be done to determine how known optimisation metaheuristics such as Tabu search may be applied in the context of the proposed framework.

Also, work has been commenced to develop a platform that can incorporate simulation, optimisation and empirically collected heuristics to affect and track architectural changes and their comparative benefits. Ultimately, the goal is to create a possibility of automatic execution of steps involved in architectural optimisation.

References

[1] G. Agosta, G. Palermo, and C. Silvano. Multi-objective co-exploration of source code transformations and design space architectures for low-power embedded systems. In *SAC '04: Proceedings of the 2004 ACM symposium on Applied computing*, pages 891–896. ACM Press, March 2004.

[2] E. Airoldi, X. Bai, and R. Padman. Markov blankets and meta-heuristic search: Sentiment extraction from unstructured text. *Lecture Notes in Computer Science*, (3932):167–187, 2006. Springer Verlag, Berlin.

[3] A. Borshchev. System dynamics and applied agent based modeling. In *In Proceedings of International System Dynamics Conference*, Boston, MA, July 2005.

[4] A. Borshchev and A. Filippov. From system dynamics and discrete event to practical agent based modeling: Reasons, techniques, tools. In *In Proceedings of The 22nd International Conference of the System Dynamics Society*, Oxford, England, July 2004.

[5] R. D. Braun and I. M. Kroo. Development and application of the collaborative optimization architecture in a multidisciplinary design environment. In *International Congress on Industrial and Applied Mathematics, Hamburg, Germany, August 1995*, August 1995.

[6] V. Campos, M. Laguna, and R. Marti. *New ideas in optimization*, chapter Scatter search for the linear ordering problem, pages 331–340. McGraw-Hill Ltd., UK, Maidenhead, UK, England, 1999.

[7] M. Denford, J. Leaney, and T. O'Neill. Non-functional refinement of computer based systems architecture. In *Engineering of Computer-Based Systems, 2004. Proceedings. 11th IEEE International Conference and Workshop on the*, 2004.

[8] J. W. Forrester. *The Systemic Basis of Policy Making in the 1990s*, chapter System Dynamics and the Lessons of 35 Years. Sloan School of Management, MIT, Boston, MA, 1991.

[9] F. Glover, J. P. Kelly, and M. Laguna. New advances and applications of combining simulation and optimization. In *1996 Winter Simulation Conference*, pages 144–152, 1996.

[10] L. Grunske. Transformational patterns for the improvement of safety properties in architectural specification. In *Proceedings of the The 2nd Nordic Conference on Pattern Languages of Programs (VikingPLoP 03)*, 2003.

[11] P. Haddawy. An overview of some recent developments in bayesian problem solving techniques. In *AI Magazine Special Issue on Uncertainty in AI*, July 1999.

[12] R. Kazman, J. Asundi, and M. Klein. Quantifying the costs and benefits of architectural decisions. In *Proceedings of the 23rd international conference on Software engineering*, pages 297–306. IEEE Computer Society, 2001.

[13] S. Kodiyalam. Evaluation of methods for multidisciplinary design optimisation(mdo), phase i. Technical Report NASA/CR-1998-208716, NASA, September 1998.

[14] S. Kodiyalam and J. Sobieszczanski-Sobieski. Multidisciplinary design optimization - some formal methods, framework requirements, and application to vehicle design. *Int. J. Vehicle Design (Special Issue)*, pages 3–22, 2001.

[15] C. Maxwell, T. O'Neill, and J. Leaney. A framework for understanding heuristics in architectural optimisation. In *13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2006)*, Potsdam, Germany, March 2006.

[16] C. Maxwell, A. Parakhine, M. Denford, J. Leaney, and T. O'Neill. Heuristic-based architecture generation for complex computer systems. In *12th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2005)*, 4th-7th April 2005.

[17] J. Mokus and W. Eddy. *Bayesian Heuristic Approach to Discrete and Global Optimization*. Springer, 1996.

[18] E. Rechtin. *The Art of Systems Architecting*. CRC Press LLC, 2nd edition, 2000.

[19] A. Trendowicz and T. Punter. Applying bayesian belief networks for early software quality modeling. *IESE-Report No. 117.03/E*, (Version 1.0), 2003. Fraunhofer IESE.

[20] I. Tsamardinos, C. F. Aliferis, and A. Statnikov. Algorithms for large scale markov blanket discovery. In *The 16th International FLAIRS Conference*, St. Augustine, Florida, USA, 2003.

[21] J. van Gurp. *On The Design & Preservation of Software Systems*. PhD thesis, Rijksuniversiteit Groningen, February 2003.

[22] J. van Gurp and J. Bosch. Automating software architecture assessment. In *In Proceedings of Nordic Workshop on Programming and Software Development Environment Research 2000 (NWPER 2000)*, Lillehammer, Norway, May 2000.

Expanding the view on Complexity within the Architecture Trade-off Analysis Method

David Colquitt[1], John Leaney[2]
1. Faculty of Engineering
2. Institute for Information and Communication Technologies
University of Technology, Sydney
[David.Colquitt | John.Leaney]@uts.edu.au

Abstract

The following paper presents the learning outcomes from an investigation into the aspects of complexity involved in architecture-based analysis. Using a framework of situational complexity as provocation, the manifestations of complexity observed in the Architecture Tradeoff Analysis Method (ATAM) process are presented in terms of a people and systems dimension. These aspects of complexity are shown to impact upon some of the most important ATAM objectives. The change in ATAM complexity is also presented with respect to the design lifecycle. Some resolution to the complexity suffered by the process is suggested in terms of splitting out the analysis objectives and maintaining two types of analysis, as well as paying attention to the content aspects of the process that drive its direction from within.

Keywords
Software architecture, architecture analysis, situational complexity, ATAM

1. Introduction

Empirical [1] and theoretical [2] work both support the notion that the further along the system design and development path, the more committed stakeholders are to the solution and the more costly it becomes to change design decisions [3]. The tangible saving in terms of both effort and money has created a decisive need to reason about the finished systems properties using a baseline of only the earliest design artefacts. Software architecture (SA) is one such discipline that seeks to make use of abstract design representations, encouraging communication amongst stakeholders and providing a vehicle for reasoning about the design from the earliest system form onwards [4]. By establishing a relationship between the goals of a system, be they functional or non-functional and the underlying structure of the system, SA suggests the capability to design for desired properties.

The need to design for specific properties also requires a method by which these properties can be tested for in architectural designs, giving rise to architecture-based analysis techniques. Architecture-based analysis developed recently during two distinct epochs, the first marked by questioning techniques oriented towards candidate selection [5], the second marked by methods more focused on analysis throughout the design process [6]. Coincident with the change in analysis role was the expansion of the participating stakeholder group, showing a shift from expert-centric to stakeholder-centric analysis. The resulting group of participants can be considered, as they are in the broader design process, a *human activity system*, granting architecture-based analysis many of the attributes of a *social* or '*soft*' process [7].

Expanding the stakeholder group heightens the situational complexity, which Flood suggest comprises a 'classic/systems' dimension and a 'people' dimension [8]. Until recently the focus has remained on the systems aspects of architecture-based analysis with only recent acknowledgement of the people dimension. "as architecture reviewers, we continually run into social, psychological, and managerial issues and must be prepared to deal with them." [9]. The different world views (Weltanschauung) [10] of each participant will naturally effect how they view the task and participate therein. Until now the responsibility for handling this aspect of complexity has been placed in the hands of the facilitator. This is notably dismissive of the need to alter the process itself and the potential for the facilitator themselves to contribute to the complexity of the situation. The following paper presents an experience-based account of researching the complexity of architecture-based analysis within a combined industry-university design project.

Section 2 presents the importance of the research methodology and the research situation to the interpretation of the outcomes, as well as introducing the chosen architecture-based analysis method. The aspects of complexity identified within the Architecture Trade-off Analysis Method (ATAM) are then presented (Section 3) and their impact on the ATAM process discussed (Section

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

4). Learning from applying the process at different stages in the design lifecycle is then discussed in Section 5 and a brief conclusion given in Section 6.

2. Research Background and Methodology

2.1 Research situation

The research was undertaken as part of a broader project whose aim was to develop a proof of concept design for a next generation network (NGN) management system. Traditionally telecommunications infrastructure has been strongly engineered for quality, meaning that a relatively static topology and offline configuration of the network by-in-large accounts for the quality perceptions of the user. However faced with a significant decline in the value of its traditional services, the telecommunications market has perceived the need to enable new value added user services. The desire to provide innovative new service sets to consumers has created a step-wise increase in the complexity of management systems and placed them in the critical path for service delivery. In many ways the business capability is now seen as being defined by the capability of the management systems [11]. This significantly augmented the quality expectations of a class of systems that were already considered in the domain of complex systems.

Consequently the quality focus of software architecture and the telecommunications problem seemed a natural fit, realising a linkage project between the university and an industrial partner who was a leading vendor of telecommunications equipment and services. The project group consisted of academics from within the university, senior technology consultants from the industry partner and several doctoral researchers. The academics were drawn from different telecommunications and software disciplines. Their roles included oversight of each of the key project discipline of Architecture, Policy, Networking and Service. Each discipline had at least one associated doctoral researcher.

The research focus of this paper is a subset of the work undertaken as part of the Architecture practice of the team. While the issues addressed in this research are pertinent to the broader practice of architecture-based design as they are the specific area of architecture-based analysis, the focus was chosen due to reasons of involvement and clarity. The researchers were heavily involved in the Architecture practice of the design team. The issue of clarity refers to the ability to clearly identify the research area. Architecture-based analysis is a contained event that occurs within architecture-based design practice with a clearly defined method. Whereas the design team were consciously not following a structured design approach due to the nature of the research challenge.

2.2 The Architecture Trade-off Analysis Method (ATAM)

Architecture-based analysis techniques fall into one of two categories, questioning and measuring according to whether they offer qualitative or quantitative results. In complex design situations the effort required to develop models suitable for quantitative analysis and the concentration on one quality at the expense of others tend to dissuade the use of measuring techniques. While many of the subsequent questioning techniques provide the ability to evaluate multiple quality aspects of a system and don't require quantitative models, they still tend to only find application as candidate selection methods once a design has reached maturity [5].

The adoption of an iterative incremental development process required a method which could be used throughout the systems lifecycle, as well as provide insight into the design issues and how they relate to the customer objectives. Consequently the methods suited to such an approach are those oriented towards application from an early point in the design life-cycle as well as providing the ability to analyse the relationship between multiple quality concerns and design decisions. The only methods found to satisfy these conditions included Software Architecture Assessment using Bayesian Networks (SAABNet) [12] and the Architecture Tradeoff Analysis Method (ATAM) [13]. Although it is viewed as qualitative in nature, SaabNet requires the numeric coding of relationships between design aspects as conditional probabilities and as such requires determinism in the relationship between design moves and system properties that is not known.

The Architecture-based Trade-off Analysis Method (ATAM) was selected as the most appropriate for the research project as it could be used throughout the design lifecycle, achieved design analysis rather than candidate selection and had a strong lineage of development backed by case reporting. The method itself is broken into two overlapping phases of 9 steps as shown in Figure 1.

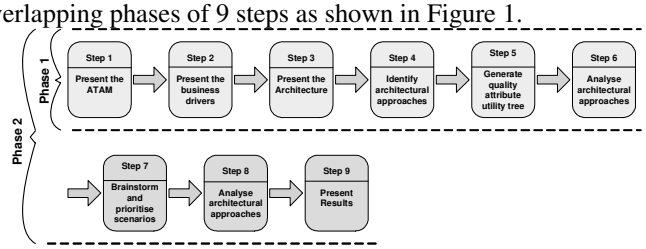


Figure 1 - Architecture-based Tradeoff Analysis Method (ATAM)

The first step introduces the method itself to the participants. This is followed by two steps presenting both the business case and the solution architecture respectively. The 4th step looks to identify key architectural approaches responsible for system qualities. The 5th step creates the attribute utility tree, which refines the business drivers into

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

quality goals into concrete scenarios representative of the goals. The final steps identify architectural sensitivity (architectural decision key to a specific quality) points, trade-offs (architectural decision key multiple qualities) and risks (important decisions not made) [13]. (need to potentially add a diagram to this).

2.3 Action Research and Interpreting the Research Outcomes

As discussed in Colquitt [6], the original research interest in the "social" dimension of complexity promoted the idea there are difficulties the process has to overcome which are only attributable to the people within a situation. That is the problems arise from the intersection between the technical and non-technical [14], the interaction of the perspective of the stakeholders with the task of architecture-based analysis. The human dimension of which urges the use of qualitative methods to capture the complexity of the phenomena [14].

The need to act as a researcher in the telecommunications research project and simultaneously research the architecture-based analysis of systems promoted the use of a methodology that would accommodate both roles. Action Research (AR) as a methodology for situated inquiry is sympathetic of the need to perform both roles and is also accepting of change as a mechanism of developing further understanding. The need for both action and learning is revealed in the structure of the methodology which in its most abstract form consists of stages of planning action, taking action and reflecting upon action. The phases form a natural cycle, in which the reflection and learning from the previous cycle influences the planning and action to be taken in the next phase, as theory and practice both inform one another [15]. The particular method adopted for applying Action Research is that of Susman [16] Figure 2.

While Action Research reports on experience it needs to be understood as a more structured approach to inquiry than recollections of past experience. The research was undertaken with a specific aim to understand the complexity of ATAM; a specific method of inquiry (Action Research) and a defined framework of ideas representing the outcomes of background research. However the structure and discipline does not grant generality of the findings, and although the issues can be extrapolated out to large systems design the outcomes would be specifically different. The research should be understood in the spirit of Schon's "science before the fact" [2]. With the outcomes providing the basis for more controlled experimentation in the future.

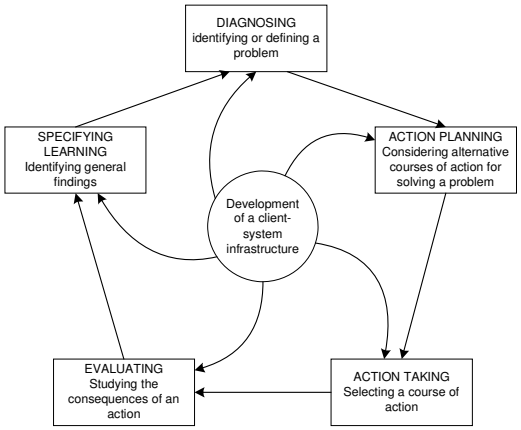


Figure 2 – Susman Action Research Model [16]

Importantly the content of the rest of this paper only seeks to present the outcomes of the research, not the processes whereby AR assisted in formulating the outcomes. Similarly for reasons of brevity some supporting material has been omitted in terms of design artefacts and quotes.

3. Complexity in the ATAM

3.1 An Expanded View of Complexity within the ATAM

The motivation for a specific focus on architecture-based analysis arose from an early project need to perform analysis on existing system designs. The research team exhibited strong diversity from the outset, and matters of design often became side-tracked by clarifying interpretations and viewpoints amongst the design group. Discussions indicated the role of the stakeholder in the project, their experience and areas of interest all contributed to the ways in which they participated. The importance of personal viewpoints, beliefs and interests aligns closely with what has been termed a 'people' dimension to complexity [8]. Importantly this aspect of complexity will increase as the perceptions and beliefs of the participants diverge.

The chosen method of ATAM promoted stakeholder participation, yet didn't openly tackle how to address the impact of stakeholder diversity. Instead the ATAM literature suggests these issues can be resolved by properly setting the expectations of the participating parties, ensuring documentation is made available and that the facilitator is sufficiently skilled [9]. The focus on the facilitation is dismissive of the need to adapt the process itself to handle such diversity. It also re-enforces the view that traditional systems engineering is focused on the 'systems' aspects of complexity at the expense of the 'people' dimension [17]. Complexity within a situation arises from both the number of parts and relationships of

the system at hand, as well as the people within the design situation. Consequently the following sections present the aspects of both 'people' and 'systems' complexity encountered within the ATAM. Section 4 will discuss how this complexity impacted what the method was trying to achieve.

3.2 People dimension to complexity

People complexity suggests each individual's capabilities, beliefs and interests will influence how they participate in design situations. These aspects of complexity formulate the perspective from which a participant views the design situation. This perspective influences the way design artefacts are interpreted and communicated amongst the group. The term Weltanschauungen (W) [10], meaning world view is commonly used in Information Systems (IS) literature to express the idea that artefacts and actions can be interpreted in different ways according to each individual's perspective. Significant evidence for differing world views was found throughout the ATAM exercises.

3.2.1 Differing world views. The second and third steps of the ATAM are designed to present the business case for the system and the system architecture. These presentations are given by the industry partner (client) and the design team respectively. The difference in language and concepts used within these presentations provided the first insight into the perspectives at work within the analysis. The business drivers presented by the client emphasised the needs of cost management, including both operational and capital measures, as well as customer choice and satisfaction. These drivers are all contingent on the way in which the system is designed but are nonetheless, quite distant from the quality attributes such as performance and security commonly put forward when talking about system architectures.

The difference in language and concepts exhibited in the presentation activities of the ATAM also carried through into the quality attribute workshops. These workshops are designed to elicit the key quality attributes of the system that form the first tier of the utility tree. The quality responses of the industry partner used telecommunications business concepts such as "network optimised" and "customer aware". Alternatively the broader group tended to re-enforce the systems quality perspective adopted by the software architecture community, offering qualities like "performance" and "availability". However rather than simply re-enforcing the dichotomy of business and systems quality, the quality responses also indicated a more personal value-based view of the quality needs for the system. Vocal advocates of standards offered qualities like "standards-based", similarly those with broader experience in billing and mobile aspects to networks raised notions like "roaming" and "billing accuracy".

The language used to define the attributes themselves also reflected the personal notions of the participants. Those with backgrounds in telecommunications management used terms such as "protocols", "managed data", "5 9's" and "Frame Loss Rate". Participants with more hands on networking experience referred to "rack space", "moves/adds/changes (MACS)". Whereas the architecture-minded amongst the group used well known architectural constructs such as "connections" and "components" [18], the extreme of which was a systems architect who clearly defined 3 qualities then offered the view that these three were defined/contributed to by the remaining qualities.

3.2.2 Influence of Roles and Beliefs on Participation. The impact of personal beliefs on how individuals exercised their roles within the analysis also became quite evident through the project. In developing group artefacts there is an inherent aim to accommodate the views of all participants present. This accommodation creates a natural tension between group consensus and personal opinion. Participation in these social situations is based upon appreciative acts. Appreciative acts concern both judgements of reality (what is the case?) and judgements of value (is this acceptable or unacceptable?) [19]. These judgements cannot be ego-less and are intrinsically linked to the participant's view of their role in the situation. There were several instances of roles and judgement shaping individual participation.

The belief that qualities can be hierarchically organised influenced the way in which the lead architect, then facilitator went about utilising the quality responses. Instead of tallying the votes, the facilitator decided to use the quality relationships to group responses in a bid to include a broader range of quality attributes rather than simply selecting the most popular. Quality attributes are a key concern of software architecture and the groupings attracted the attention of a member of the architecture team, who sought to change the end result of the exercise.

Similarly in building the utility tree, the lead designer raised issue with assumptions about the system creeping into scenarios. Being the designer, any aspects of design that crept into the development of artefacts would directly impact their work and potentially force decisions that they themselves were unwilling to make at that stage. On each of these occasions, the impact of group processes on personal roles prompted remedial action by a participant.

3.2.3 Negotiating of Meaning as Critical. Given the diversity of language and concepts discussed in section 3.2.1 it is not surprising that the negotiation of meaning was an important theme throughout the research. Open dialogue helped surface assumptions and grow each

participant's understanding. However this was seldom a controllable, brief exercise. Discussions of systems quality commonly branched into other areas of design and into business strategic considerations. Meaning was rarely just about semantics but a roundtrip through an associated network of concepts.

The hard won nature of negotiated meaning means that it is strongly defended. On separate occasions different participants sought to augment the set of quality concerns with a quality they saw as important following the completion of the quality attribute workshops. While the group could see the importance of these attributes they had not passed through the same process of group negotiation and couldn't be viewed as representative of the group's viewpoint. In the instance where changes were made to artefacts outside of the group processes, there was a significant degree of scepticism towards the changes. Some more damaging changes resulted in disinterest within the broader group.

3.3 'Systems dimension' to complexity

Whilst the initial focus of the research was the effect of the people dimension on complexity, there was commonly found to be contributing factors from the systems dimension. Be it due to the intractable nature of design, the conceptual nature of the system at such an early stage of design or the various ways a system can be decomposed,, it became apparent that behind most people problems, complex systems aspects could be seen to be co-incident.

3.3.1 Concomitant nature of the problem and solution. One of the most prominent aspects of systems complexity in the ATAM proved to be the relationship between the business-strategic and system requirements perspectives. Exploration of either perspective seemed to require knowledge from the other in order to understand it. For example considerations of the impact of specific quality requirements on a system commonly reverberated back to considerations at the business strategic level. Similarly in attempting to resolve answers to questions at a business strategic level, knowledge of the capability of the system was often sought. The problem and solution appear to evolve together and become concomitant. In software design the learning loop is perceived as taking place between the requirements and the design artefacts [20]. While this is indeed necessary and true, experience here has shown that the requirements embody an approach that attempts to resolve a business need for the client. Therefore the loop of learning between the original motivating problem and the approach lies as much between the aspirations of the client and the driving requirements of the system as it does between requirements and design activities. Potentially it is even more critical at this stage since the loop of learning bridges world views as well as from problem to solution (as section 3.2.1 attests).

3.3.2 Divergent nature of understanding. The difficulty in reconciling these viewpoints lies in developing a complete understanding of them. The search for solutions cannot be exhaustive due to sheer number of permutations in complex systems [21]. Experience from a goal-based requirements (GBR) workshop to bridge the business strategic perspective and systems quality perspectives helped highlight this aspect of complexity. Several goal-graphs of up to eighty nodes were produced, which only represented the higher level considerations.

3.3.3 Difficulty developing usage aspects. Another of the consistent difficulties in attempting to communicate aspects of the solution or problem was the elusive nature of use, or how the system would be used. Early on in the design lifecycle the system architecture is incomplete, hindering attempts at understanding the potential usage aspects of the system [22]. Even in the event that a complete functional structure was to be available there is still some doubt as to whether this adequately reveals the context of use [23].

Additionally no real precedent for such a telecommunications system existed. Therefore notions of use which would develop from detailed system knowledge were unclear. In their place abstractions of use, in this case the operational aspects, or operational specifications, were put forward. The problem with operational specifications like those so commonly used in telecommunications is that they are solution agnostic. They specify what has to be done but don't give clarity on how it should be achieved. A task like the ATAM really requires the structural detail behind how things are achieved to understand the quality ramifications. Two systems could quite readily exhibit the same operational characteristics but have two entirely different systems (structurally, architecturally) implementing them.

Further obstructing the understanding of usage was an expectation of innovation. The project has been conceived with the intention that the NGN management framework would supersede existing management practises. As such traditions and experience became largely invalid because they were perceived as coming encumbered with the past mistakes.

3.3.4 Environmental Turmoil. Complicating matters was the speed with which important environmental influences could change. Telecommunications is an rapidly evolving industry where technology and carrier behaviour is constantly changing. From within the organisation there were multiple company acquisitions and new patents brought to the design situation. The social, political and technological forces influencing the project made it

difficult to stabilise the linkage partner's position, exacerbating the difficulty understanding the strategic and system quality associations.

4. Impact of situational complexity for the ATAM process

4.1 Disconnect between the business strategic and systems quality perspectives

Where aspects of systems and people complexity discussed in section 3 are coincident upon a process it is understandable they would affect the conduct and outcomes. Perhaps the most enduring of these affects was the difficulty to associate and understand the business strategic and systems quality perspectives. The relationship and understanding between these perspectives is fundamental to the aims of what the ATAM is designed to achieve. [4].

The ATAM literature offers a fairly close relationship between these two informational elements, in many instances proposing what are more commonly recognised as quality attributes, as business drivers. "For example, in an e-commerce system two of the business drivers might be stated as: "security is central….and modifiability is central to the success of the system…"" [13]. Contrary to these examples, the earlier discussion of the 'people dimension' to complexity established these as two different perspectives (world views). Importantly the actors aligned with these perspectives are likely to use significantly different language and concepts to express the driving system need. The diversity of these viewpoints caused difficulty for the ATAM in two main ways. The first was in understanding and utilising the responses. The second was building the business goal to system quality relationships necessary for constructing the utility tree.

4.2 Common understanding of quality viewpoint

The ATAM literature states the potential mismatch in communications resulting from the business owner and designers having to communicate through an intermediary in the form of a facilitator [9]. When confronted by the relatively unique quality terms of the linkage (industry) partner the facilitator sought to interpret them in more popular systems quality terms. The perspective was an important one, yet was unlikely to receive much representation in the process while it differed so significantly from the broader group. The experience affirmed the legitimacy of the systems quality perspective and the alignment of the facilitator's world view with it. Instances like this where participant's views are challenged in the face of broader quality frameworks have the potential to ostracise participants from a process where their input is critical.

Conducting the exercise from a specific world view can also stifle more diverse viewpoints. Or more worryingly suppress a unique insight which has not yet been appreciated by the rest of the group. Early on in design situations the problem is ill-structured and the more creative solutions challenge the brief rather than fulfil it [21]. The ATAM may be the first occasion stakeholders have to express specific insights they have to each other. Participants commonly showed significant interest in each other's qualities when they were tabled.

The constructive dialogue shows the importance of negotiating meaning [24], which should precede any negotiation of objectives or goals. Negotiating meaning is something that is only lightly raised in the ATAM. Although it professes the need for facilitators to seek concurrence and feedback [9], there is little suggestion as to what this is. Meaning in terms of the ATAM is primarily "negotiated" through the construction of the utility tree, where ambiguity of meaning is resolved through concrete scenarios. As discussed earlier in this section, by this stage the intended contribution of the participant may have been significantly altered to conform to quality attribute norms. Similarly scenarios proved no refuge for understanding as early on the usage context of the system is not well understood and interdependencies with business strategic issues can hamper the development of more detailed design concepts. This represents nothing of the social framework in which the negotiation of meaning should take place [24].

By not openly negotiating meaning throughout the process, ATAM tends to leave the participants isolated in their own perspective, rather than permitting them the ability to evolve their understanding of the situation with respect to each others views. This can significantly enhance the analysis process in both the emergent outcomes from constructive understanding as well as the less tangible aspects of providing the group with a common identity and understanding. The 'taken as given' meaning of individual perspectives [10] in contrast to group consensus was perhaps no better exemplified by a glossary exercise undertaken between analyses, in order to define problematic terms. It took several workshops and significant effort before the group was able to agree upon a definition for the word "service" alone. Notably the time taken to establish such meaning is not looked upon kindly where project schedules are important.

4.3 Disjoint between systems quality and business drivers

4.3.1 Assumptions in bridging perspectives exemplified by Quality of Experience. While the systems quality and business strategic perspectives proved to be

quite different, the utility tree requires causal attributes be made between them in order to focus the analysis. Characterisations could be attempted in order to draw relationships between the business and quality viewpoints. However assumptions generally have to be made in order to do so. For example in the NGN solution space, systems commonly refer to the need for high levels of throughput and performance to ensure customer satisfaction [25]. Yet there are no guarantees that a performing system will be the determining factor in the customer's view of the service. The customer may be happier with a low performing cheaper service, or a service delivery method that does not have any real-time implications. The concept of Quality of Experience (QoE) acknowledges that the customer does not just use technology but lives with it [26]. Consequently quality aspects associated with the usage perspective partially influence, but aren't solely responsible for the customer perception [27]. Care needs to be taken in testing the assumptions behind framing the problem in a particular way [2], which is effectively what these characterisations are inviting participants to do. Viewing the customer satisfaction as largely a network performance issue narrows out of view other contributing factors to customer experience like ubiquity, cost, cultural appeal, etc [28].

4.3.2 Exhaustive Understanding of Requirements is Infeasible. Attempts at understanding the relationship between the systems quality perspective and business strategic perspective are complicated by the concomitant nature of understanding between them. Section 3.3.1 outlined how understanding of the business and technical solutions were dependant upon one another. Seeking to resolve the problem by reaching an exhaustive understanding of either perspective is likely to be infeasible given the combinatorically explosive nature of search through the solution space [21], exemplifying the divergent nature of real world situations [2]. Modelling the customer goals highlighted this divergence showing that quality attributes were implicit in some of the goals but their subsequent refinement provided no guarantees of yielding explicit quality attributes akin to those commonly found in software architecture literature. Furthermore the depth of reasoning (some 6-8 layers of hierarchy) highlighted how much refinement logic was internalised by the goal to quality associations.

4.3.3 Analysing against systems quality does not necessarily ensure customer satisfaction. The extent and complexity of the strategic viewpoint and the problematic nature of characterising goals as system qualities calls into question whether the system as judged from the quality viewpoint, satisfies the business strategic goals. From the experience of this research, this is largely not the case. Although the ATAM addresses important quality concerns there is no certainty these concerns ensure the satisfaction of the customer's business objectives or allays their greatest worries. The lack of certainty affect's the confidence of the group when deriving the attribute utility tree, which is effectively the centrepiece of the analysis. When faced with a utility tree devoid of business context, the participants sought to elaborate the utility tree with aspects of the business drivers. On reconciling the business and systems quality aspects, apprehensions were expressed as to the rigour of the relationship. The associations were re-analysed and changed according to the new consensus. The ready acceptance of change highlighted how unconvincing the original relationship was in that the group was comfortably able to reason through many changes.

5. ATAM and the design lifecycle

5.1 Design stages and their affect on the ATAM process

The previous sections presented the elements of situational complexity found to affect the ATAM and discussed the consequences for the process. The exercises that contributed to this learning all occurred across a significant time frame of the project. This granted insight into the use of ATAM both early on and throughout the design lifecycle. The following reflects learning on the relationship between ATAM and the design lifecycle.

Early in the design lifecycle quality requirements generally represented broader issues within the business drivers. The dependency of the system qualities on the business objectives saw a strong focus on clarifying the customer goals. The rationale for these goals appeared to be most strongly influenced by the external environment of the business. The constant pace of technological change and the ability for systems to define market capability continuously challenge the business to redefine itself in line with its environment[29]. These environmental factors, as well as scarring past experience are largely what the customer brings to the design situation [21]. Additionally the personal perspectives of the stakeholders ensure there is any number of views of the situation early on. At this stage in design the stakeholders are largely conceptually isolated and the focus of early activity is on negotiating meaning.

Once meaning has been negotiated the group can meaningfully discuss the quality aspects of the system in a bid to move from the business strategic to more systems design considerations. Personal meaning starts to become associated with group beliefs as negotiated artefacts are developed in group situations. Although in this design situation showed these beliefs were still confused by the lack of clarity surrounding the shape of the system design and its usage context. Importantly this experience only braces the early conceptual stage of design and successful ATAMs recounted within existing case reporting indicate

that once the design is of sufficient maturity, usage concepts become embodied along with detailed behavioural understanding of the system [30]. The detail of the design helps reveal usage aspects and focuses the group on technical challenges, testing against what are considered to be fixed notions of strategic direction.

Figure 3 attempts to depict the progression of the design and the associated characteristics of analysis situation. Boehm's spiral model [31] is used as its overlay to indicate the early stages of design close to the origin and the later stages of design towards to the outer layers.

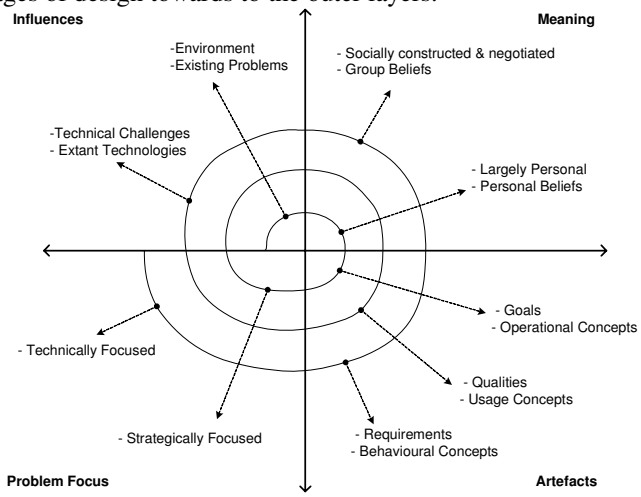


Figure 3 - Stages of design and its influence on group process

Evidenced by the experiences throughout this project as depicted in Figure 3 , the use of ATAM early on is problematic for several reasons. The first is that the ATAM does not focus on making sense of the business strategic viewpoint. Rather it is more focused on using it to develop a list of business drivers. However there are business problems to be solved as much as there are technical ones. Secondly it seeks to move quickly from the business strategic viewpoint into the systems quality one, which is actually a significant evolution in design terms. Lastly emanating from the quick progression from the business perspective to the quality one, there is an assumption of group understanding. The difficulty is that this has neither been given the open forums in which it needs to develop, nor the structure of detailed artefacts upon which it is commonly built.

The ATAM relies on well reasoned and stable understanding of the business strategic needs. Problematically these needs are exposed to the constant turmoil of the external environment and tend to develop in concert with, rather than well ahead of the system design.

Ideally the concomitant nature of the problem and the solution supports the need to deal with both at the same time. An approach that could potentially permit analysis at an early stage is to tailor the process according to the stage of the design. Figure 3 conveys the idea there are different needs at different times in the design lifecycle. Initially the problem focus is strategic; the strongest influences are previous system failures and changes in the business environment; meaning within the group is largely experience/belief-based and personal; and the system is represented by abstract artefacts such as goals and broad operational concepts.

Dealing with modelling the system goals exhibited very similar traits to system qualities, in that they were generally abstract, needed refinement to be properly understood and exhibited interdependencies, akin to design trade-offs.. These similarities imply the potential for an early ATAM exercise focused on the use of goal-level artefacts to establish the consequences between different strategic approaches. This would yield significant knowledge on business strategic issues permitting the customer stakeholder sufficient, rather than partial representation in the process. It also helps model the customer problems that tend to consistently upset the technical design process. The knowledge in this area could then be used to feed into the more technically focused ATAMs once the system has developed significant maturity, which sees the design stakeholders as receiving sufficient representation. Separating these two types of analysis allows the customer to analyse business issues with the same rigour that systems analysis covers design issues. The outcomes of the business focused analysis would help provide a firm basis upon which to build the quality requirements for the systems analysis. The business analysis also provides a rigorous mechanism to confront the constant change in the external environment, rather than continuously exposing the system design to the business uncertainty.

5.2 The importance of 'content' versus 'process'

In addition to the need to consider two different but inter-related types of analysis during the design lifecycle, there also needs to be a greater consideration for what analysis activities are achieving. In looking at design methodology Dorst noted that although most methodologies were specified in terms of processes, it was largely the "content" of the design situation that dictated the designer's actions. "In most cases, considerations linked to the content of the design situation (the perceived design problem, the designer's goals and the perceived possibilities for the next step) will determine the 'kind of action' (process-component)" [32].

It is therefore not surprising that the reality of systematic design conflicts with the way in which it is prescribed [21]. Similarly the ATAM concentrates heavily on the process itself at the expense of the content issues, which can significantly affect an analysis. For example is the goal of the presentation of the business context just that? Or is it to reach agreement and understanding within the group of the

strategic issues influencing the system development and to carefully derive a set of system characteristics that can be seen to satisfy the strategic needs. Not forgetting of course there are multiple perspectives here. Do the designers have few expectations of the business context presentation and participate simply as a passive audience? Or do they expect a comprehensive presentation of issues availed to them through their interaction with the client, which they can rigorously question and seek to understand in a detailed way?

Where the reality of the activities does not fulfil the expectations of a stakeholder, there is little guidance for how resolution may occur. In this project improvisation took the form of glossaries, elicitation of meaning where it was not required, goal-quality matrixes and extensive elicitation of goal artefacts, amongst other things. Although explicit methods were used here to explore the improvisation there is a distinct danger that in commercial settings, with greater time pressures, where the facilitator would feel pressured to maintain in "control" of the exercise, the improvisation may well take the form of internal judgement and assumption. The danger of which is a self-sealing process [33], whereby the internal assumptions of the participants are not tested and any incidents that lurch towards difficulties in understanding are avoided lest they hamstring the entire process. However it must be acknowledged that this research can only point to the potential for this to occur due to the experience in the linkage project.

6. Conclusion

The ATAM represents a significant evolution for architecture-based analysis techniques. In place of masking the analysis process in the problematic scoring and manipulation of figures with a perceived end of candidate selection, the ATAM has taken responsibility for assisting understanding throughout design. The inclusive nature of the process also ensures that communication amongst the stakeholder community is enhanced. However as discussed in this paper the inclusive nature of the process and the conceptual nature of architectures challenges the evolved methods, particularly early on in the conceptual stages of design. The resultant situational complexity impedes some of the key objectives that the ATAM seeks to achieve, such as improved communication and a relationship between systems quality and stakeholder goals.

The impacts on the process can be seen to arise from the structure and perspective of the ATAM and notably extend well beyond the scope of facilitation. The process itself needs to adapt in order to provide the social framework in which the negotiation of meaning and objectives takes place. Currently the diversity of viewpoints are more likely to smooth over diversity than to openly encourage it into the process. However the uncertainty the richness of such diversity was found to impact on the design task which struggles to progress when exposed to constant change.

Consequently two streams of analysis have been proposed as a means of isolating the design perspective from the constant change of the business strategic environment. Further enhancement is also proposed through a greater focus on the content aspects of the system, which will drive participation from within the analysis aside from the external structure of the process imposed from outside it.

7. Acknowledgements

Funding for this work was furnished in part by the Australian Research Council (ARC) in the form of an Australian Postgraduate Award (APA) scholarship for David Colquitt. Additional funding and support has also been made available through an ARC Linkage Grant (grant number LP0219784), and the industrial partner, Alcatel Pty Ltd..

8. References

1. Brooks, F.P., *The mythical man-month : essays on software engineering*. Anniversary ed. ed. 1995: Addison-Wesley Pub. Co. xiii, 322 p.
2. Schön, D.A., *The reflective practitioner : how professionals think in action*. 1991, Aldershot, England: Arena. x, 374.
3. Abowd, G., et al., *Recommended Best Industrial Practice for Software Architecture Evaluation*. 1997, Sofware Engineering Institute (SEI), Carnegie-Mellon University (CMU): Pittsburgh, Pennsylvania.
4. Bass, L., P. Clements, and R. Kazman, *Software architecture in practice*. 2nd ed. SEI series in software engineering. 2003, Boston: Addison-Wesley. xxii, 528.
5. Lee, J. and K.-H. Hsu, *Modeling software architectures with goals in virtual university environment.* Information and Software Technology, 2002. **44**(6): p. 361-380.
6. Colquitt, D. *The case for understanding social complexity in the architecture-based analysis process*. in *Qualitative Research in IT & IT in Qualitative Research*. 2004. Griffiths University: Griffiths University.
7. Jagodzinski, P., et al., *A Study of electronics engineering design teams.* Design Studies, 2000. **21**: p. 375-402.
8. Flood, R.L., *Complexity: a Definition by Construction of a Conceptual Framework`*. Systems Research, 1987. **4**(3): p. 177-185.
9. Kazman, R. and L. Bass, *Making architecture reviews work in the real world.* Software, IEEE, 2002. **19**(1): p. 67-73.
10. Checkland, P. and S. Holwell, *Information, systems, and information systems : making sense of the field*. 1998, Chichester ; New York: Wiley. xv, 262.
11. Hope, M. and E. Nichols, *OSS: New Approaches for Successful Service Management*. 2002, OVUM.
12. van Gurp, J.B., J. *SAABNet: Managing qualitative knowledge in software architecture assessment VO -*. in *Engineering of Computer Based Systems, 2000. (ECBS 2000) Proceedings. Seventh IEEE International Conference and Workshopon the*. 2000.

13. Kazman, R., M. Klein, and P. Clements, *ATAM: Method for Architecture Evaluation*. 2000, Software Engineering Institute: Pittsburgh.
14. Seaman, C.B. and V.R. Basili. *An Empirical Study of Communications in Code Inspections*. in *Proceedings of the 19th international conference on Software engineering*. 1997. Boston, Massachusetts, United States: ACM Press.
15. Avison, D., et al., *Action Research.* Communications of the ACM, 1999. **42**(1): p. 94-97.
16. Susman, G.I. and R.D. Evered, *An Assessment of the Scientific Merits of Action Research.* Administrative Science Quarterly, 1978. **23**: p. 582-603.
17. Jackson, M.C., *Systems methodologies as complementary tools for managing situational complexity.* Transactions on Instrumentation Measurement and Control, 1988. **10**(3): p. 155-160.
18. Perry, D.E. and A.L. Wolf, *Foundations for the Study of Software Architecture.* SIGSOFT Softw. Eng. Notes, 1992. **17**(4): p. 40-52.
19. Checkland, P., *Soft Systems Methodology: A thirty year retrospective.* Systems Research and Behavioural Science, 2000. **17**: p. S11-S58.
20. Fischer, G., D. Redmiles, and L. Williams, *Beyond Object Oriented Technology: Where Current Approaches Fall Short.* Human-Computer Interaction, 1995. **10**: p. 79-119.
21. Joseph, S., *Design systems and paradigms.* Design Studies, 1996. **17**: p. 227-339.
22. Brooks, F.P., *No Silver Bullet: Essence and Accidents of Software Engineering.* Computer, IEEE, 1987. **20**(4): p. 10-19.
23. Kroes, P., *Design methodology and the nature of technical artefacts.* Design Studies, 2002. **23**(3): p. 287-302.
24. Sargent, P., *Design science or nonscience.* Design Studies, 1994. **15**(4): p. 389-402.
25. Bouch, A., M.A. Sasse, and H. DeMeer. *Of Packets and People: A User-centered Approach to Quality of Service*. in *Eighth International Workshop on Quality of Service. IWQoS 2000*. 2001. Piscataway, NJ, USA: IEEE.
26. Keinonen, T. *User experience with communication technology*. in *Telework*. 2001.
27. Sheridan-Smith, N., J. Soliman, and D. Colquitt. *Improving the user experience through adaptive and dynamic service management*. in *Australian Telecommunications Networks and Applications Conference*. 2004. Bondi Beach, Sydney.
28. Armbruster, H., *Information infrastructures and multimedia communications: different approaches of the telephone, data, and radio/TV worlds.* Communications Magazine, IEEE, 1997. **35**(9): p. 92-101.
29. Truex, D., R. Baskerville, and H. Klein, *Growing systems in emergent organisations.* Communications of the ACM, 1999. **42**(8): p. 117-123.
30. Clements, P., R. Kazman, and M. Klein, *Evaluating software architectures : methods and case studies*. SEI series in software engineering. 2002, Boston: Addison-Wesley. xxv, 323.
31. Boehm, B.W., *A spiral model of software development and enhancement.* Computer, 1988. **21**(0018-9162): p. 61-72.
32. Dorst, K. and J. Dijkhuis, *Comparing paradigms for describing design activity.* Design Studies, 1995. **16**(2): p. 261-274.
33. Argyris, C., *On Organizational Learning*. Second ed. 1999, Massachusetts: Blackwell Publishers. xv 464.

An Event-driven Architecture for Fine Grained Intrusion Detection and Attack Aftermath Mitigation

Jianfeng Peng, Chuan Feng, Haiyan Qiao, Jerzy Rozenblit
Department of Electrical and Computer Engineering
The University of Arizona
Tucson, AZ 85721-0104, USA
jr@ece.arizona.edu

Abstract

In today's computing environment, unauthorized accesses and misuse of critical data can be catastrophic to personal users, businesses, emergency services, and even national defense and security. To protect computers from the ever-increasing threat of intrusion, we propose an event-driven architecture that provides fine grained intrusion detection and decision support capability. Within this architecture, an incoming event is scrutinized by the Subject-Verb-Object multipoint monitors. Deviations from normal behavior detected by SVO monitors will trigger different alarms, which are sent to subsequent fusion and verification modules to reduce the false positive rate. The system then performs impact analysis by studying real-time system metrics, collected through the Windows Management Instrumentation interface. We add to the system the capability to assist the administrator in taking effective actions to mitigate the aftermath of an intrusion.

1. Introduction

Network attacks are a fundamental threat to today's largely interconnected computer systems. Most of these attacks share the same characteristics: intrusion into computer hosts that have securities holes [1]. Unauthorized accesses and misuse of critical data on intruded hosts not only cause loss to personal users, but also pose a threat to the entire corporate network because in many cases the intruders use the compromised nodes to launch larger scale attacks. Firewall and antivirus packages are often insufficient in detecting and preventing all intrusions, especially attacks from insiders [2]. Efficient Intrusion detection systems thus are needed to form another important line of defense in the face of increasing vulnerability.

In the past, different types of IDS have been proposed and built. However, a study of currently existing IDSs reveals that most operate at a coarse grain level [3]. For example, Steven et. al. proposed an approach that uses sequence of system calls to identify potential threats [4]; Warrender proposed a similar approach [5]. Ghosh utilized return address information extracted from the call stack to generate an execution path for a program to detect anomaly [6]. While these IDSs use different feature representations of system calls, they treat events as an integral part and are unable to investigate internal characteristic such as executers, event objects, etc [7]. This directly affects detection accuracy. Another shortcoming of the coarse grain IDS is that reaction is only available after an event is completed, thus the system is unable to preempt a harmful operation before it completes.

To achieve more efficient intrusion detection, we need a more insightful understanding of the system's ongoing events. Fine grained event checking capability is an essential part of the architecture proposed in this paper. It is implemented using Subject-Verb-Object multipoint monitors. Any event that is taking place in the system is modeled using SVO structure. Alarms are triggered with respect to each element of the triple to achieve fine grained anomaly detection. The proposed architecture employs two databases that maintain metadata per user. The user metadata provides a basis for detecting deviation from normal user behaviors. The system metadata records real-time system performance metrics to facilitate alarm verification and impact analysis.

This paper is organized as follows: Section 2 briefly describes the SVO techniques used to model system events. Section 3 presents the proposed architecture. Section 4 discusses issues that are related to some of the function modules. Section 5 provides the DEVS Simulation results and some concluding remarks.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

2. Event Modeling Using Subject-Verb-Object Triple

In order to study the internal characteristics of an incoming event, we need a formal way to denote the following aspects: event executer, operation, and event object. In linguistic typology, such a structure is commonly called an SVO triple. Using this pattern, we can model an ongoing event with a triple that contains all the detailed information we are interested in.

2.1. The SVO Triple

Any event happening inside the computer host has its subject, which most of the time is a running process. For instance, typing a letter is often associated with a text editor; redrawing a client area is initiated by the parent window, while a *put* command in a sftp session usually comes from the ftp program. Similarly, we can describe the verbs and objects for these three events. Verbs of an event tell us what type of operation is performed on the objects. Objects of an event are normally hardware (peripheral devices, ports, etc) or software resources (files, drivers, libraries, etc) on the computer. At a higher level, these processes are owned by the current user, whose behavior is modeled by studying all the SVO triples when he is using the computer on a daily basis. By putting the subject, verb and object into a triple, we have a formal structure in the following way:

{ *MFC Window,*	*Redraws,*	*Client Area*	}
{ *Text Pad,*	*Reads,*	*MyFile.txt*	}
{ *FTP Program,*	*Opens,*	*Port 4567*	}
{ *User Program,*	*Writes to,*	*Serial Port*	}

…

We collect information of ongoing events using custom developed software tools. By dissecting an event into these three fields, we are able to perform fine grained analysis of the events. One may question the efficiency of this modeling technique as there are virtually infinite numbers of combinations that can happen when the computer is running. This problem is addressed by limiting the events to be scrutinized to a finite set of critical subjects, critical verbs and critical objects. Any event outside of this set will either pose no threat to the system or be a trivial threat that can be ignored. Fortunately, most of the events happening every moment on a computer do not belong to the critical event set (CES), thus we can focus on studying the behaviors of the ones that do belong to the critical event set as described in the following section.

2.2. The User-Configurable Critical Event Set

Since every user on a particular computer host has unique access privileges to resources, it is computationally very expensive to define a critical set that works for every user on every computer. Instead, it is necessary for the IDS to allow customizable critical set for each user. Depending on the nature of the data that reside on the computer and the actual role of the computer, this critical set can vary from a simple set that contains a few password protected files, to a much more comprehensive set of all files on C drive, local ports, and even hardware resources. In the current stage of our research, we focus on the critical set containing important files that need protection. The CES is defined as: {*, *, D:\EmployeeFiles*.doc}.

This critical set mandates that the IDS needs to monitor any process that attempts to perform any operations on any .doc files in the D:\EmployeeFiles directory. Consequently, any operation that is not accessing the data contained in that directory is ignored under the current configuration.

2.3. Adding Time Information

Each SVO, when saved into the user database, is tagged with a timestamp. The purpose of the timestamp is twofold. First, it adds an additional dimension to the event model and allows us to gain better understanding of the abnormality of current events. For instance, a particular file access operation is deemed normal during regular work hours; however, it is abnormal out of regular work hours. Secondly, timestamp makes it possible to perform temporal alarm fusion, which helps to reduce duplicated alarms.

By combining time information with an SVO triple, we have a mechanism that allows us to explore the following questions in a fine grained manner: who does what, to whom, at what time.

3. SVO Based Intrusion Detection Architecture

With the SVO modeling technique and custom defined CES described in section 2, we propose an architecture as depicted in Figure 1 to perform fine grained checking on all incoming events. When a new event comes in, it is intercepted by the IDS that runs in the background and sent into a multistage monitor. The monitor investigates the subject, verb and object by comparing them respectively to the normal behavior stored in the user database.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

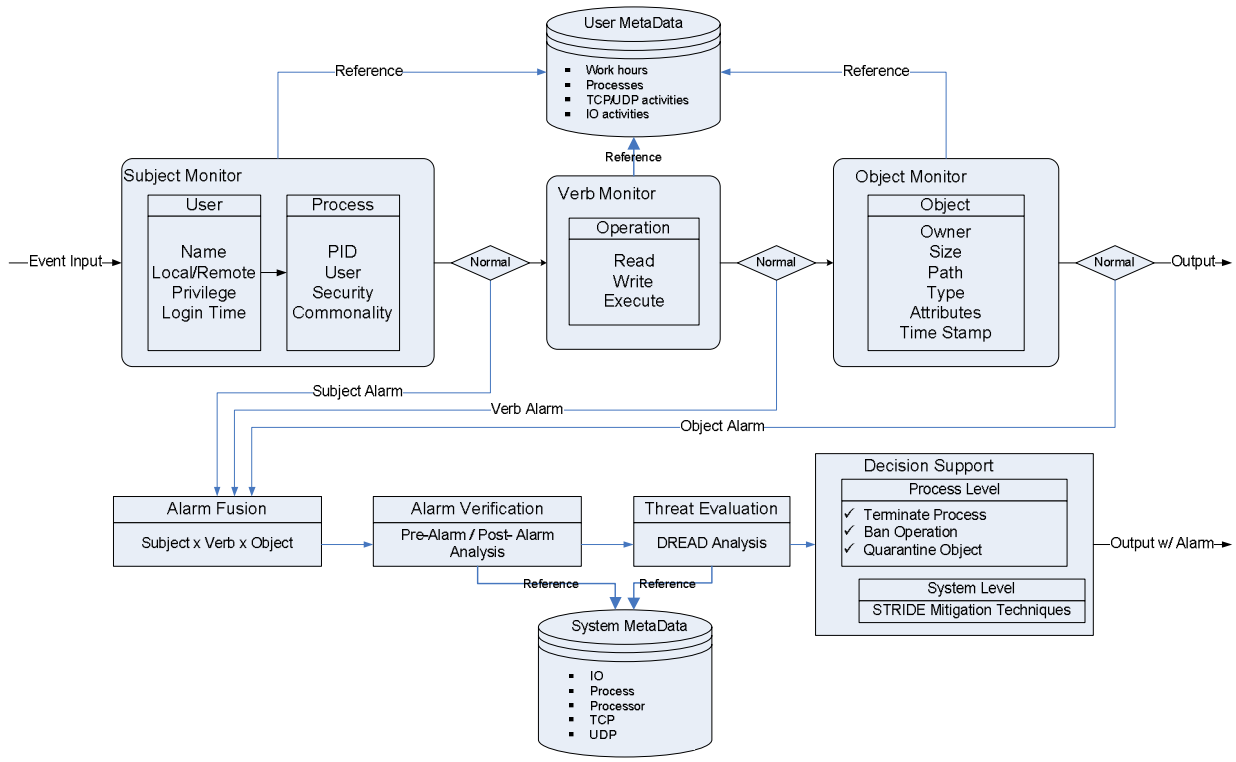


Figure 1 The Event-Driven Architecture

The user metadata database contains information that tells the IDS at what time which processes will normally take what kind of actions on what objects. A new event will trigger an alarm if it involves objects defined in the Critical Event Set, and if any element of the SVO triple deviates from normal behaviors. For instance, in our sample setting, all *.doc files in D:\EmployeeFiles directory are protected. Now if a process tries to modify a *.doc file in that directory, an Object alarm will be triggered first. The same event may also trigger Subject Commonality alarm if that process has never been observed before inside the current time window. In this case, the event violates multiple security rules and triggers multiple alarms, which are to be fused and verified subsequently.

The SVO based architecture has the following advantages. First, it doesn't require a priori knowledge of intrusions as needed in signature based [3] or system call traces based approaches [4]. Second, intrusion can be detected at the earliest time possible because the IDS does not need to wait until the operation has successfully been executed or damage has been caused. Third, an event is investigated at fine grain level so that more appropriate and effective reactions can be taken according to the nature of the intrusion.

Figure 1 shows all the essential components of the proposed architecture. These include the SVO based anomaly detection engine, alarm fusion module, verification module, threat evaluation module and decision support module. These modules address such problems as how to determine if an event is abnormal, how to deal with an alarm, what reactions should be taken when a threat is confirmed. The remainder of this section discusses the functionalities of these modules in more detail.

3.1. Anomaly Detection based on User Profiling

Intrusion detection techniques of IDSs can be categorized into two classes: signature-based and anomaly-based. Signature-based IDSs compare current events with known attacks and look for similarities, such as comparing a sequence of system calls to known attack patterns. This method has the major limitation of not being able detect novel attacks [3]. To overcome this, our proposed architecture uses an anomaly-based detection approach, which models normal behaviors and attempts to identify abnormal activities on the computer.

The precondition for such an anomaly detection IDS to work is to create a range of normal behaviors. In our system, this normal base is established through

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

an extensive user profiling process whose purpose is to build an in-depth knowledge of how the user uses this computer. The following information is derived: during normal usage of the computer, what processes take what kind of actions on what hardware resources or software objects?

To achieve this, we developed a software tool, SysMon as shown in Figure 2, which uses WMI to retrieve runtime system metrics. SysMon combines information from Spy++ event logs and records the results into a user profile database. More details regarding the database can be found in section 4.1. To model normal host behavior, both supervised and unsupervised learning algorithms can be applied.

Unsupervised learning algorithms take as input a set of unlabeled data and attempt to find intrusions contained in the data. It can be treated as a variant of the classical outlier detection problem and does not require the input data set to be fully normal. Outlier based anomaly detection algorithms cluster the data based on certain metrics and the points located on sparse regions are treated as intrusions. The unsupervised algorithms make two important assumptions about the data which motivate the general approach. The first assumption is that the number of normal instances dominates that of abnormal instances. The second assumption is that the abnormal instances are qualitatively different from the normal instances. The basic idea is that since the anomalies are both are rare and different from normal, they will appear as outliers in the data and thus be detected [8]. The clustering process scans through the data collected by WMI, and identifies normal instances and outliers.

From the results of the above mentioned clustering process, we have a clear knowledge of the user's normal usage of the computer. For instance, one can conclude that User A on the monitored computer has the following normal behavior with regard to file read operations in D:\EmployeeFiles directory during the regular work hours of *9am* to *5pm*:

Winword.exe,	*File Read Operations/sec,*	**.doc*	*[50,200]*

...

The above generalization gives an accurate indication of how a user normally uses his computer. Thus once a new event occurs, the anomaly detection engine will compare it to the existing profile of that user and determine whether the event falls into the range of normal behavior. Once a deviation is detected, the detection engine raises an alarm.

3.2. Alarm Fusion

Once an abnormal event happens, it is very likely to trigger multiple alarms at Subject, Verb and Object checkpoints. In order to minimize redundant alarms and condense alarms that stem from the same event into one integral alarm, an alarm fusion module is needed.

In the proposed architecture, a multi-level alarm fusion algorithm is used. The first one is a source preprocessing level, which synchronizes the information flow from different sensors to reduce data redundancy for further processes. The second level is alarm normalization, which transforms different alarms into a consistent set of scale. The third level is spatial alarm fusion, which fuses alarms from different anomaly detection monitors. The fourth level is temporal alarm fusion, which analyzes alarms within a certain time window and gives more useful intrusion information. Further details of the fusion mechanism can be found in [9]. After alarms are fused, they are sent to the verification module.

3.3. Alarm Verification

The task of this module is to verify the correctness of fused alarms in an effort to reduce false alarms. The verification module works by checking the normality of an event that has triggered an alarm. This includes checking the frequency of similar operations that have been performed before, as well as identifying the security level of the objects that the process is trying to access. While these are automated processes, the verification module also provides a human-computer interface that allows the system administrator to participate in the decision making process.

3.4. Impact Evaluation

A successful attack on a computer usually results in considerable impact on some aspects of its normal operation. Such impact includes disruption of critical services, undermined computation capability, increased network latency due to excessive outbound traffic and hardware resource exhaustion. To evaluate the impact of an attack, it is necessary to carry out a detailed comparison of system characteristics before and after an alarm is triggered. This is made possible by our real-time system performance monitoring tool, which collects system run-time performance data and saves them into a system metadata database as shown in Figure 1. The impact evaluation module queries the database about the following system metrics to find any differences before and after the event:

- Number of running services
- Processor time, queue
- Memory usage
- Network bandwidth, latency

3.5. Decision Support and Aftermath Mitigation

One of the major contributions of the proposed architecture is its capability to provide insightful information on current attacks and more precise counteraction with regard to the Subject, Verb or the Object. Depending on which elements of the SVO triple pose threats to the protected system, the decision support engine can automatically perform any one or a combination of the following three categories of actions:

- Ban current user
- Terminate operation
- Quarantine objects

In addition to the above three instantaneous actions, the decision support engine uses a rule based reason system that will investigate the nature of the attack, and take post-attack actions to eliminate security vulnerabilities and prevent the same type of intrusion from happening again. For instance, security level of the objects can be escaladed immediate to prevent future unauthorized access. A system administrator will be notified automatically. The administrator will often perform further investigation into the nature of the event and install patches to eliminate security holes. Howard proposed a variety of security precautions in building secure software [10]. Many of those techniques can be applied to unaffected systems on the same network to prevent the propagation of the attack.

The architecture also provides an interface that can report the current status to a network based fusion engine. The network based fusion engine collects information from each computer node, and correlates the data to perform higher-level situation analysis. Information provided by the interface includes everything that is needed to determine at what time, which user, by which process, is taking what kind of action against which object. With such fine-grained information, the network based fusion engine is able to take more effective actions to mitigate the aftermath of an intrusion such as terminating established TCP connections, closing ports, and isolating the affected host.

3.6. Performance Analysis

Once the IDS is installed on a computer, it runs constantly in the background to protect hardware/software resources defined in the CES. This adds to run-time computational overhead similar to antivirus packages. The overhead depends on the size of the CES. A smaller CES will have less impact on the system performance than a larger CES because the latter will involve checking on more events in run time. On computer systems that contain sensitive data, advantages of the SVO based IDS will greatly outweigh potential performance impact. The impact can be further reduced by introducing a security screening process that checks login attempts by any user out of normal operation hours.

4. Design Issues: Collecting User/System Data through WMI

The architecture proposed in this paper heavily relies on the capability to collect real-time information at both the system level and the process level. Microsoft WMI is a set of extensions to the Windows Driver Model that provides an operating system interface through which instrumented components can provide information and notification [11]. WMI includes real-world manageable components, available from the DMTF standards with some specific extensions that represent the various Windows components. To locate the huge amount of management information available from the CIM repository, WMI comes with a sql-like language called the WMI Query Language (WQL). WMI can be used to obtain data about your hardware and software by writing a client script or application, and data can be provided to WMI by creating a WMI provider.

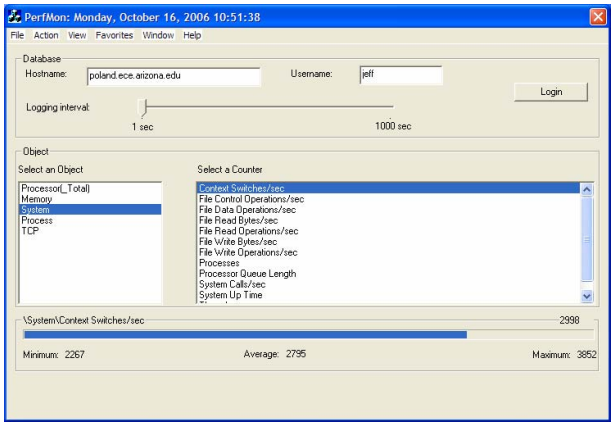


Figure 2 Real-time User Profiling & System Performance Monitoring Tool (SysMon)

Figure 2 is a screenshot that shows the SysMon tool we developed to collect system run-time information through WMI. The following is a snapshot of the information collected using the real-time user profiling and system performance monitoring tool that runs on our desktop systems:

ID	User	Usage	Counter	CurTime
11	1	1332	Processor(_Total) Interrupts/Sec	2006-09-20 17:08:04
12	1	16	Processor(_Total) Processor Time	2006-09-20 17:08:35
13	1	0	Processor(_Total) User Time	2006-09-20 17:08:59

…

5. Summary

To verify the overall effectiveness of the proposed architecture, a simulation platform has been built to simulate attacks and study the reactions of the IDS system. The simulation uses Discrete Event System Specification (DEVS) [12] to describe the proposed architecture. The simulation is described in detail in [7]. The initial experimental results are very promising. We are currently obtaining real world data on which a full verification of the proposed approach can be carried out. Our future research will focus on the mitigation of attack aftermath.

References

[1] B. Schneier, "*Attack Trends 2004 and 2005*", ACM Queue vol. 3, no. 5 - June 2005

[2] R. Chinchani, A. ILyer, H.Q. Ngo, S. Upadhyaya, "Towards a Theory of Insider Threat Assessment", Proceedings of International Conference on Dependable Systems and Networks, 28 June-1 July 2005 Page(s):108 - 117

[3] S. Axelsson, "*Intrusion detection systems: A survey and taxonomy*", Technical Report 99-15, Department of Computer Engineering, Chalmers University, March 2000.

[4] S. A. Hofmeyr, S. Forrest, A. Somayaji, "*Intrusion detection using sequences of system calls*", Journal of Computer Security, Volume 6, Number 3, 1998.

[5] C. Warrender, S. Forrest, B. Pearlmutter, "Detecting intrusions using system calls: alternative data models", Proceedings of the 1999 IEEE Symposium on Security and Privacy, May 09-12, 1999.

[6] A. K. Ghosh, A. Schwartzbard, M. Schatz. "*Learning program behavior profiles for intrusion detection*", In Proceedings of the 1st USENIX Workshop on Intrusion Detection and Network Monitoring. USENIX Association, April 11-12 1999.

[7] A. Liu, C. Martin, T. Hetherington, Sara Matzner, "A Comparison of System Call Feature Representations for Insider Threat Detection", Proceedings of the Sixth Annual IEEE Systems, Man and Cybernetics (SMC) Information Assurance Workshop, 15-17 June 2005.

[8] H. Qiao, J. Peng, C. Feng, J. Rozenblit, "Behavior Analysis-Based Learning Framework for Host Level Network Intrusion Detection", to be published in Proceedings of the 2007 IEEE International Conference and Workshop on the Engineering of Computer Based Systems, March 2007.

[9] C. Feng, J. Peng, H. Qiao, J. W. Rozenblit. "Alert Fusion for Intrusion Detection and Decision Support", to be published in Proceedings of the 2007 IEEE International Conference and Workshop on the Engineering of Computer Based Systems, March 2007.

[10] M. Howard and D. LeBlanc, "Writing Secure Code," Microsoft Press, 2nd edition, December 4, 2002.

[11] Microsoft, WMI: Introduction to Windows Management Instrumentation, http://www.microsoft.com/whdc/system/pnppwr/wmi/WMI-intro.mspx

[12] B.P. Zeigler and H.S. Sarjoughian, "Introduction to DEVS Modeling and Simulation with JAVA: Developing Component-Based Simulation Models," Draft Version 3, 2005

Using Weak Bisimulation for Enterprise Integration Architecture Formal Verification – I

Dr. Ernest Cachia, Mr. Mark Vella
University of Malta, Msida, Malta
ernest.cachia@um.edu.mt, mvel002@um.edu.mt

Abstract

Weak Bisimulation is a process calculus equivalence relation, applied for the verification of communicating concurrent systems [26]. In this paper we propose the application of Weak Bisimulation for Enterprise Application Integration verification. Formal verification is carried out by taking the system specification and design models of an integrated system and converting them into value passing CCS (Calculus of Communicating Systems) processes. If a Weak Bisimulation relation is found between the two models, then it could be concluded that the EI Architecture is a valid one.

The formal verification of an EI Architecture would give value to an EI project framework, allowing the challenge of cumbersome and complex testing typically faced by EI projects [18], to be alleviated, and thus increasing the possibility of a successful EI project, delivered on time and within the stipulated budgeted costs.

This paper shows the applicability of value passing CCS (or equivalent) formal notation to model the EI systems characteristics, as well as investigates into the computation complexity of available weak bisimulation algorithms, in order to analyze the applicability of this proposition in real life.

1. Background

In the process of searching for an Enterprise Integration (EI) specific framework to guide an integration team in the strategic implementation of an integrated IT landscape within and beyond the scope of a single enterprise, an extensive research in the following areas of Enterprise Integration was made: -

- Integration and middleware technology [7] [20] – for acquiring the understanding of the technological mechanisms that make systems integration possible.
- Standards [3][20] [28] – to be aware of the agreed upon standards in order to work on their lines.
- Scientific foundation [26] [29] [28] – in order to be able to add value to the field of Enterprise Integration based on the concepts of Computer Science and Software Engineering.
- Challenges [9] [10] [18] [20] [21] [22] [33] – in order to locate those Enterprise Integration specific areas that need improvement.
- Methodology and Best Practices (Linthicum 2003) [30] [33], [32] – in order to have the knowledge of existing improvement efforts and possibly build on them.

The industry research, made up mainly of compiled industry reports, articles [4] [16] and literature [5] [7] [10] [20] [30], allowed the broad understanding of the current state of EI projects in industry; from the projects business drivers, technologies and methodologies being used, to the factors affecting the success of these projects. On the scientific level, the research investigated which areas of computer science and software engineering could be applied to EI projects, in order to improve the situation of this area. [26] [29] [28]

1.1. Software engineering principles

A sound software engineering framework is one that delivers high quality software deliverables on budget and on time. [11] [29] In the case of an EI-specific framework, it is being proposed that in addition this would be a framework that is targeted specifically at EI systems, that achieves the EI-specific goals and qualities, allowing the EI project challenges to be overcome, and thus maximizing the probability of success and avoiding project failures as identified in [21].

1.2. EI-specific framework value

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

The proposed value of an EI-specific Project Framework could be better explained in the following scenario: take a software project manager who has managed traditional software projects for some time, but is now faced with the challenge of setting up an EI project plan. He/she should be aware of the fact that managing an EI project, although still a software project, requires a specific management framework to address the specific EI challenges. An EI-specific management framework would be very beneficial, in this particular case, to start building the EI project plan and carrying out all the necessary tasks leading to an effectively built EI system.

1.3. EI project challenges, goals and qualities

Further to what was presented in [9], according to [21] and [12] the main challenges causing failure in EI Projects include:

- Lack of standard methodologies – so far only industry best practices and EI product specific methodologies were found.
- Lack of proper business process definitions – in fact many business models today exist only in the head of departmental managers and at times these also conflict with the understanding of their colleagues.
- Lack of business units co-operation – In several cases, business units only communicate to put the blame on each other, and compete fiercely for company budgets. On the other hand, EI projects require full business unit co-operation.
- Implementation is more complex than expected – An EI implementation usually consists of several implementation technologies, packages from different vendors, multiple platforms and an unexpected number of interface links.
- Relying too much on integration technology for project success – Middleware technology in fact is only an enabler of integration implementation but far from being a complete software engineering tool.
- Lack of thorough testing - this is so, given the newly introduced integration level and the enterprise wide scope of such systems.
- Lacking the proper integration team roles – EI projects, given their novel nature, are only seen by the IT department as just another IT project, and fail to re-organize the IT roles before the start of these projects.

[30] compiled a list of goals and qualities that should be reached/exhibited by an EI-specific project framework. These are as follows: -

(Goals)
- Ensure that the EI architecture and developed applications satisfy business needs
- Describe how to manage the EI process
- Describe how to work with legacy systems and packaged solutions to integrate them
- Provide guidance on technology selection and standardization
- Ensure that the methodology promotes reuse

(Qualities)
- Align IT with the enterprise business strategy
- Build on a solid enterprise architecture
- Leverage legacy and commercial software
- Focus on security

In addition to these goals and qualities, the main strategic business value of EI today is that of being an enabler of Business Process Management [23] [24] [19]. This point was taken into consideration and a decision was taken to focus on the Business Process Integration [10] type of integration, where the main focal points of integration are the business process and not the applications. Here, Application Integration is only a consequence of joining up the business processes, but not the main driver.

These goals and qualities along with the EI project challenges form the basis for the reasoning underlying an EI-specific framework.

1.4. Framework building blocks

The foundation of this framework is made up of building blocks from the fields of Computer Science, Software Engineering and Business Management. These building blocks are: -

1.4.1. Value passing CCS (Calculus of Communicating Systems) or equivalent Process Calculus – from the field of Computer Science that allows the mathematical modeling of communicating concurrent and mobile systems. [26] CCS allows the formal specification and reasoning of communicating concurrent systems. Its applicability to the EI domain is shown in section 3 of this paper.

1.4.2. Unified Modeling Language (UML) – a software engineering tool allowing the modeling of software specification and design. [28] UML was

chosen due to the wide adoption in the software engineering world and its OMG standard status.

1.4.3. Business Process Management (BPM) – a business management discipline born out from Business Re-engineering that advocates end to end business process modeling, automation and their continuous monitoring and optimization. [5] This building block was made part of the project in order to be able address the goal of EI systems to be an enable to Business Process Management programmes.

This paper concerns only the application of the first building block: value passing CCS. More specifically, Weak Bisimulation, that is a binary relation on CCS processes [26], is being proposed as a possible tool for the verification of EI architectures.

1.5. Scope

The whole process required for completing the formal verification of an EI architecture is illustrated in figure 1, where the business process model defines the system specification and the EI Architecture is the system design that is verified against the business process model.

This paper shows the applicability of process calculus such as value passing CCS to the domain of EI, present a treatment of Weak Bisimulation and the possibilities this offers as a formal verification tool, and place the verification step within the context of an EI-specific framework. The formal specification and design, as well as an in-depth look at the verification process will be treated in subsequent papers.

2. Formal EI verification proposition

Weak Bisimulation (≈) is a process binary equivalence relation based on the equivalence of just the observable reactions between 2 processes. In other words, as long as the second process can match each observable reaction sequence of the first process and vice versa, the 2 processes are regarded as weak bisimilar, irrespective of their internal reactions. Thus Weak Bisimulation is also known as observation equivalence, with the processes in question regarded as black boxes

[26] shows how the weak equivalence relation (≈) could be used to prove that a particular system structure implements correctly a particular system definition. More specifically he showed the application of Weak Bisimulation as follows: ***System ≈ Specification.***

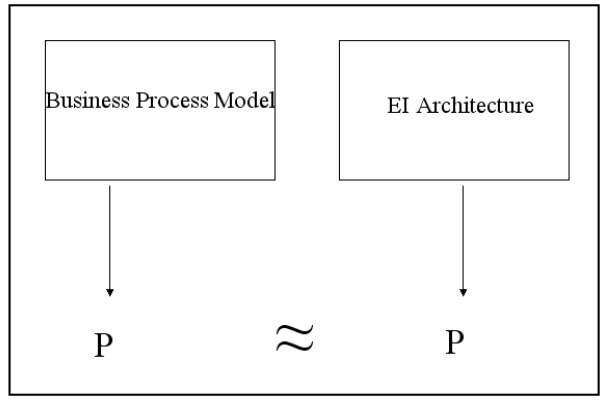


Figure 1 – EI Architecture verification process

Weak bisimulation is the chosen process equivalence relation for the fact that a system specification does not have as yet an internal structure and as a consequence, minimal internal reactions. Thus, this reasoning rules out both Structural Congruence (≡) and Strong Bisimulation (~). Given that Weak Equivalence (≈) is insensitive to both internal reactions and structure [26], it fits well the need of proving the correctness of the implementation of a system against its specification.

Applied to the domain of Enterprise Integration, or more specifically to the chosen Business Process Integration type of integration, this observation equivalence relation is being proposed to be applied as follows: -

Business Process Model ≈ EI Architecture

The Business Process Model defines the required business process flows to be automated by the underlying system, whilst the EI Architecture is the design of the EI system implementing the business flows. By mapping these models into value passing CCS processes, the equivalence between the Business Process Model and the EI architecture, could be formally verified by finding a Weak Bisimulation relation between these two processes.

3. EI systems characteristics

From the initial research underpinning this paper, the following architectural characteristics of EI systems stood out: -

- Concurrent Systems [7] [20]

- Business Process Modelling [GB and Ruh 04] [McGov01]
- Mobility [Smit and Fin 03]
- Security [Ruh 00]

The next four sub-sections introduce these characteristics and show how value passing CSS fairs, in modeling these characteristics.

3.1. Concurrent Systems

In EI architectures, concurrency is exhibited by the several applications, services and middleware executing in parallel, whilst messaging is the communication link between them. In Enterprise Integration, messaging is carried out by several middleware technologies that link applications together. [20] categorizes the middleware technology available today as follows: -

- Remote Procedure Calls – this type of middleware allows a software process to make synchronous calls to remote processes. E.g. Java Remote Method Invocation (RMI) [17]
- Message Oriented Middleware – this type of middleware is a queuing software that allows software processes to write and read messages to and from a queue. Communication between processes is asynchronous with guaranteed delivery. E.g. MQSeries. [14]
- Distributed Object Transactions – this is a middleware infrastructure allowing the exposure of business logic making up applications, supported by a transactional platform. E.g. Component Object Model (COM) [25] and Common Object Request Broker (CORBA) [6]
- Database Oriented Middleware – this kind of middleware provides software processes with access to database servers. E.g. Open Database Connectivity (ODBC) [15]
- Transaction Oriented Middleware – this type of middleware provides co-ordination of information movement and method sharing within the scope of a transaction. These are mainly to link legacy procedural applications to the transactional enterprise level. E.g. Tuxedo [2]

As defined by [26], value passing CCS – a formal way of modeling concurrent communicating systems, where variables are allowed along communication channels - is able to model concurrency and messaging, in the following ways: -

3.1.1. Concurrency. Being an extension of the CCS (Calculus of Communicating Systems) [26] value passing CCS supports the modeling of concurrent processes with the construct $P ::= (P1 \mid P2)$, where processes *P1* and *P2* are parallel composed together.

3.1.2. Messaging. The following value passing CCS constructs support the modeling of messaging as represented in Business Process Models (Figure 2) and EI Architectures (Figure 3). Figure 4 shows how these are represented in value passing CCS.

Input channel - $x(y).P$, means input a name on channel x by substituting with place holder y, and use the input in process P. In the EI scenario, x can represent a listening port such as a web service. The name y represents the place holder for an incoming message. Continuing on the example of a web service implementation, this incoming message can be an input XML (eXtendible Markup Language) document to the web service (a tagged data document), whose schema is referenced in the web service WSDL (Web Service Definition Language), which defined the interface of the particular web service.

Output Channel - $\overline{x}\langle y\rangle.P$ means output the name y on the channel named x, and then do P. In previous the web service analogy, this represents the consumption of the web service, where an XML document y is sent from the client application along channel x. In this case x is the TCP/IP based connection using the SOAP protocol.

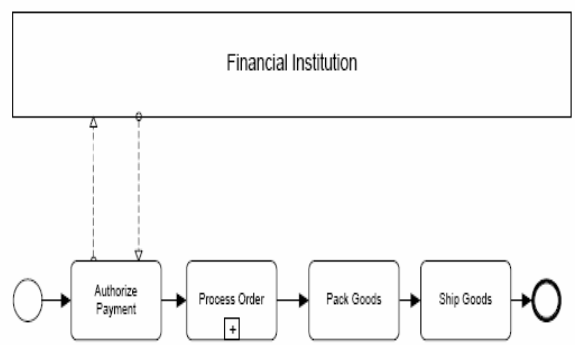


Figure 2 – Business Process Modeling Notation – Sending a message

In EI, messaging can be either **synchronous** for example in the case of a web service call or a Remote Procedure Call (RPC), or **asynchronous**, as in the case of message queues. It is possible to model both these types of messaging using value-passing CCS as follows.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

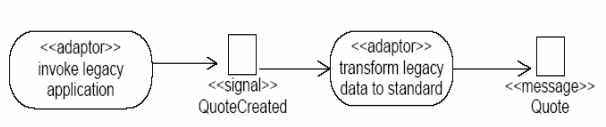


Figure 3 – Messaging in UML Activity Profile for EAI (Enterprise Application Integration) [28]

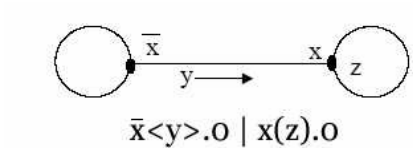


Figure 4 - Messaging in value passing CCS

3.1.3. Synchronous Messaging

$P = new\ x\ (P1 \mid P2)$

$P1 = \bar{x}\langle y\rangle.x(z).P1'$

$P2 = x(a).\tau.\ \bar{x}\langle b\rangle.P2$

Using the web service analogy in the above simplified process; an application process *P1* consumes web service *P2*. *P1* calls *P2* along channel *x* and passes the XML document *y* as an input. When the call is made, *P1 -> P1''* transition occurs, where *P1'' = (z)x.P1'*. In the *P1''* state, the calling process is kept blocking waiting on the same channel for web service *P2* to return. Once a message is returned back along channel *x* the calling process goes into state *P'* executing the rest of the process.

From the server's perspective *P2* listens indefinitely on channel *x*. When an XML document arrives along channel *x*, the web service executes its internal logic, represented by the tau (τ) symbol, and returns an output document XML back on channel *x* to *P1* on completion. The web service then resumes in state *P2* listening indefinitely for the next call.

3.1.4. Asynchronous Messaging

$P = new\ x\ y\ (P1 \mid P2 \mid P3)$

$P1 = \bar{x}\langle y\rangle.P1$

$P2 = x(w).\tau.\bar{y}\langle w\rangle.P2$

$P3 = y(z).\tau.P3$

Process *P2* handles continuous asynchronous communication between processes *P1* and *P3*. Even though the communication between *P2* and the other processes is synchronous in terms of readiness of processes to be able to communicate between each other, process *P2* provides the mechanism for asynchronous messaging between *P1* and *P3*. In the above-simplified example, *P2* models a messaging queue-like structure, allowing application *P1* to asynchronously call *P3* without blocking, even in cases when *P3* is not ready to communicate.

3.2. Business Process Modeling

Business Process Modeling involves the modeling and documentation of the business process flows within an enterprise and is a main element of Business Process Management [5] [3] [Smit & Fin 03]. EI systems are expected to serve as an infrastructure to these modeled processes, possibly by means of a Business Process Management System. [23]

CCS abstracts the notion of a process and is not specific to any particular software process living in a computer memory. Thus, it can be argued that CCS could also be suitable for modeling processes in the business sense. In the business context, we have business processes, embodying workflows, running in parallel communicating between each other inter-departmental and business to business (B2B) messages. In [34], Smith and Fingar elaborate in full detail of how Pi Calculus, an extension of CCS incorporating mobility, perfectly suites the modelling requirements of business processes.

As a matter fact, Process Calculi are already being applied to BPM for other reasons. The Business Process Modelling Notation (BPMN) adopted by the Business Process Management Initiative (BPMI) [3] is fully based on Pi Calculus foundations; as is the Business Process Execution Language For Web Services (BPEL4WS) by Microsoft Corporation [25] and IBM [13] and the BPML (Business Process Modelling Language) by the BPMI.

3.3. Mobility

From personal experience in enterprise integration projects, the communication links between the nodes of an EI architecture (applications, services and middleware) are not of fixed nature but rather of a dynamic mobile nature, where communication channels are created, moved and destroyed. For example we have the scenarios where communication channels between two processes are created as in the discovery of a web service by UDDI (Universal

Discovery Description and Integration) protocol [35]. There are also situations of channel proliferation where for example an application server or middleware becomes unavailable. There are also situations of pure mobility where a communication channel is relocated in the process space as in the scenario where a client application is instructed to start communication with an alternate server for example, for performance reasons or during a seamless, no down time, new application roll out.

Mobility in the context of business processes is reflected in the continuous change in business rules as a result of a Business Process Management Programme [Smith and Fingar, 2003]. An example is where in a move to eliminate bureaucracy an electronic components manufacturing company consolidates approvals of component designs in one department. In this case a link between the engineering department and the first auditing department in line, moves to a link with the newly created consolidated department

Pi Calculus supports the modeling of process mobility by definition; in fact Pi Calculus was invented by extending the CCS to support mobility. Mobility is modeled in Pi Calculus by allowing names representing communicating channels to be themselves passed as messages along channels. [26]

Even though Pi Calculus would be required to model mobility, for the time being only value passing CCS is being considered in order to make the proposition of this verification tool clearer. Moreover, the researched Business Process Modeling Notation [3] does not include the concept of mobility within the graphical notation; thus making value-passing CCS sufficient for formally representing business models built using this notation for the time being.

3.4. Security

Now that the applications have been 'opened up' in order to communicate possibly with the whole world, the issues of security breaches increase [Ruh, et al, 2000]. An improper security infrastructure can invalidate an otherwise well built integrated architecture. In fact the role of secure messaging increases immensely in such architectures. The importance of security in EI projects has already been identified in the Secure Application Integration Methodology by [30].

The notion of restriction in CSS denoted by *(new x) P* means that *x* is for the exclusive use of *P*. A more specific example is *P = new x (P1 | P2)* which means that *P1* and *P2* communicate via a private channel x even when placed in the context of other concurrent processes having a channel with the same name [26]. This construct enables us to model secure communication channels in integrated architectures.

Having said that, this does not mean that by using the restriction construct, it means that model is definitely secure, it only specifies that the indicated channel of communication should be secured. A typical case where a secure channel does not imply complete overall security is shown in the following system: -

$System = new\ x\ (P1 \mid P2) \mid P3$

$PI = \bar{x}\langle outval\rangle$

$P2 = x(inval).\bar{y}\langle inval\rangle$

$P3 = y(inval)$

Where even if *P1* and *P2* communicate via a secure channel, over which *P3* can never interfere, it is still up to the implementation of P1 and *P2* to ensure that that any sensitive data is not passed on to *P3*. In the above example, *P2* is sending the data received over the secure channel *x* to *P3*.

Still, having the individual processes formally defined, these can be individually formally verified in terms of system security.

4. Enterprise architecture verification

It is being proposed that CCS based verification occurs within an EI framework as follows.

Once the Business Process Model diagram (the system specification), and the EI architecture (system design) are produced, these are converted to CCS processes. This way the model passes from a semi-formal representation of the system to a formal one. If a Weak Bisimulation relation were found between the two processes, the EI architecture would be considered equivalent in behavior to the Business Process Model, and thus validated.

This verification step is carried out during the framework stage where the EI Architecture is completed and the lower level steps of design and implementation are about to start (Figure 5). Having the architecture validated at such an early stage maximises the success of the EI system by alleviating the challenge of EI testing as discussed in [18] and [21].

5. Practical applicability

In order for the proposed EI Architecture formal verification tool to be practical and usable in real world applications, the verification process must be automated, and the algorithm automating the process should do this in an efficient manner.

[1] explain that Weak Bisimulation can be decided in a time complexity of $O(n^{2.3})$, where n is the number of states, using a technique called the partitioning/splitter technique. Thus, using this algorithm, it is possible to efficiently decide a Weak Bisimulation between two processes given that: -

1. We keep the number of states finite
2. Possibly minimizing the number of states

The first condition can only be adhered to by not using variables from infinite domains (the algorithm does not work on symbolic labeled transition systems). This might sound too restrictive at first, but if it is kept in mind that at this stage the system is simply being described rather than specified in terms computations, it can be viewed from a different aspect. For example, consider a task that receives an integer, and decides on two alternative flow branches based on the value of the integer received. In the system specification and design models, one simply needs to pass a boolean value based on the size of the integer. In this case, instead of using a variable from an infinite domain, one can use a variable from a finite one. The latter can be achieved by assuming synchronous parallelism as described in [1], and thus avoiding a state size explosion when a system is made up of several parallel composed processes.

An alternate approach would be that of using a Weak Bisimulation algorithm that functions on symbolic transition systems, allowing the use of variables from an infinite domain. [8] present an efficient Bisimulation algorithm stating that is possible to port to the symbolic case.

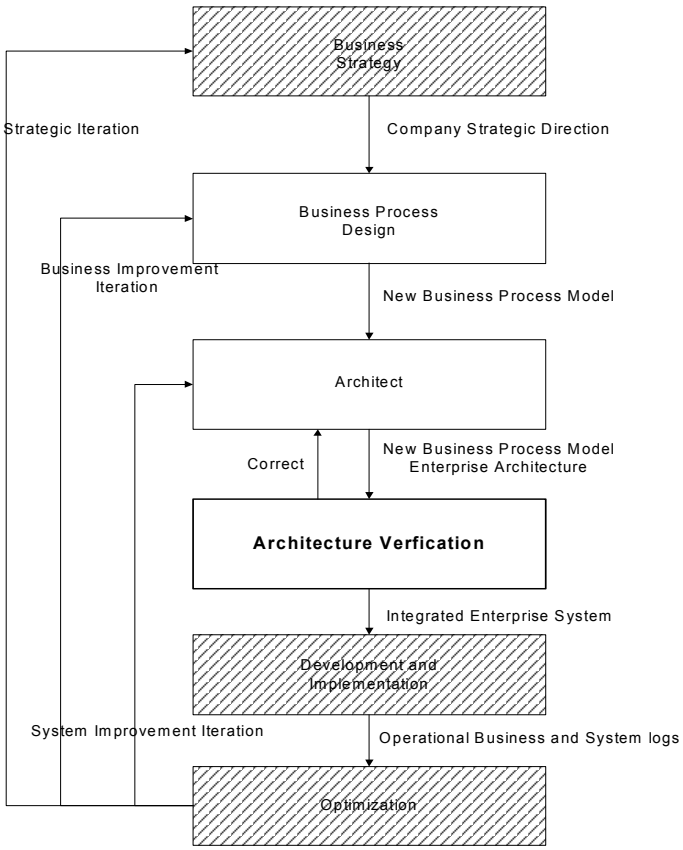


Figure 5 – Architecture Verification as part of an EI framework

6. Conclusions

This paper attempts to show that the application of value passing CCS and Weak Bisimulation as a means of validating an EI Architecture against the Business Process Model is possible by the treatment of the mapping of Business Process Models and EI Architectures to value passing CCS processes. CCS is able to model the main features of EI systems; these being Concurrency, Business Process Modeling, and Security. On the other hand the Weak Bisimulation relation is an equivalence relation that is insensitive to internal structure and reaction. This enables the checking for an equivalence relation between the Business Process Model and the EI Architecture. Whilst Pi Calculus would be required to model the remaining EI characteristic, mobility, this is not considered for the time being since mainly since no Business Process Modeling Notation researched so far addresses mobility, and also for reasons of better focusing on the usage of the proposed formal verification system in real life.

The formal verification of the EI Architecture is proposed to give value to an EI-specific project framework, allowing the challenge of complex testing typically faced by EI projects, to be overcome by allowing the formal verification of the architecture, before development and testing starts.

Finally the paper also proposed a sound foundation for a way forward for a practical application of the proposition, by mechanically automating the verification process.

7. References

[1]Baier, C. and Hermanns, G. (1999) Weak Bisimulation for Fully Probabilistic Processes

[2]BEA Tuxedo. *http://www.bea.com/framework.jsp?CNT=index.htm&FP=/content/products/tux* http://www.bea.com

[3]BPMI.ORG The Business Process Management Initiative Homepage http*://www.bpmi.org*

[4]Business Integration Journal Online. *http://www.bijonline.com*

[5]Burlton, R. (2001). Business Process Management: Profiting From Process. SAMS

[6]CORBA.ORG. The OMG's CORBA Website. *http://www.corba.org*

[7]Cummins, F.A. (2002). Enterprise Integration: An Architecture for Enterprise Application and Systems Integration. Wiley

[8]Dovier A., Piazza, C. Policriti A. An efficient algorithm for computing bisimulation equivalence.
[11][Erk & Pen 98] Eriksson, H.E. and Penker, M. (1998). UML Toolkit, Wiley Computer Publishing.

[9] Garimella, K. Ph.D. (2001). Integration Challenges in Mergers and Acquisitions. In *EAI Journal Aug 01.*

[10]Gold-Bernstein, B. and Ruh, W. (2004). Enterprise Integration: The essential guide to integration solutions, Addison-Wesley.

[11]Ghezzi, C. et al. (2002). Fundamentals of Software Engineering (2md edition). Prentice Hall.

[12]Herrera, J. (2004). Avoiding Common EAI Implementation Missteps, LogicCurve.

[13]IBM.COM IBM Homepage http*://www*.ibm.com

[14]IBM WebSphere MQ. http*://www*.ibm.com

[15]IODBC.ORG. Platform Independent ODBC. *http://www.iodbc.org*

[16]Integration Consortium *http://www.wwintegration.com*

[17] Java Remote Method Invocation (Java RMI) java.sun.com/products/jdk/rmi.

[18]Khanna, R. (2005). Top Challenges in Integration Projects. Wipro Technologies White Paper.

[19]Krishnan, M. (2004). The EAI Paradigm Shift, WIPRO Technologies White Paper.

[20]Linthicum, D. S. (2003). Next Generation Application Integration: From Simple Information to Web Services. Addison Wesley.

[21]Lublinsky, B. and Farrel M. Jr. (2002). Top 10 Reasons Why EAI Fails. In *EAI Journal.Dec' 02*

[22]Maverick, G. (2003). EAI Project Management. In *EAI Journal Nov' 03.*

[23]McGoveran, D. (2001). BPMS Concepts, Enterprise Integrity. In *EAI Journal Jan '01*

[24]McGoveran, F. (2003). Managing Business Process for EAI, In *Business Integration Journal Sep '03.*

[25]Micosoft.com Microsoft Corporation Homepage *http://www*.microsoft.com

[26] Milner, R. (1999) Communicating and mobile systems: the Pi-calculus. Cambridge
University Press.

[28] Object Management Group (2004) *http://www.omg.org*

[29]Pressman, R.S. (1996). Software Engineering: A Practioner's Approach. 4th Edition. McGraw-Hill.

[30]Ruh, A. W., et al; (2000). Enterprise Application Integration: A Wiley Tech Brief. Wiley

[32]Schmidt, J. (2003). EAI Lifestyle Evaluation. The Software Ecologist Column*,* In *EAI Journal Apr' 03*.

[33]Sifter, C.J. (2001). Integration Project Management 101. In *EAI Journal March' 01.*

[34]Smith, H. and Fingar P. (2003). Business Process Management: The Third Wave. Meghan-Kiffer Press.

[35]UDDI.ORG The Universal Description, Discovery and Integration (UDDI) protocol homepage *http://www.uddi.org*

Design and Description of a Classification System Framework for Easier Reuse

Sérgio Lopes, Adriano Tavares, João Monteiro and Carlos Silva
University of Minho, Guimarães, Portugal
<sergio.lopes,adriano.tavares,joao.monteiro,carlos.silva>@dei.uminho.pt

Abstract

Frameworks are an important technology to reduce software development costs and shorten the time-to-market. However, framework complexity presents reuse problems that limit its success as an application development alternative. It has been widely recognized that is necessary to effectively communicate frameworks and provide appropriate tool support, but we argue that difficulties are also related to the considered framework model.

We find that the level of support that tools can provide is decisively dependent on the kind of available framework documentation, which in turn is intimately tied to the considered framework model. In this paper, we present our framework model and respective supporting notation. We describe their appliance to the design and description of an application framework. This case study illustrates how the proposed model and documentation approach alleviate difficulties of framework reuse, namely its comprehension and instantiation.

1. Introduction

In an ever-increasing computer dependent business world, software becomes more complex and a crucial part of the financial plan. Therefore, the development costs and time-to-market must be reduced as much as possible, while ensuring its qualities and correctness. Frameworks, components and product lines are the dominant industry practice technologies to achieve the aforementioned long-time pursued goals of software engineering. These technologies are not mutually exclusive, and frameworks are often used along with the other two.

A framework is an incomplete design that factors commonalities and variabilities of a specific domain of application. It provides implementation for the ready-to-use common elements (*frozen spots* in [1]) and localizes variability at predefined abstract variation-points (*hotspots* in [2]). Framework design is more abstract and complex than usual programs and the dependencies between collaborating elements can be indirect and obfuscate the architecture (the structure of the framework components and their interactions with the environment)[1].

Consequently, frameworks are often hard to understand. The slow and possibly large learning curve is an important limitation for the successful adoption of framework-based reuse. Other problems exist from framework development to its maintenance, as identified in [3], but this work is especially concerned with the complications frameworks pose to application developers. It has been widely recognized that it is necessary to effectively communicate frameworks and provide appropriate tool support, in order to minimize the effort required to build applications. Several steps have been given in this direction (e.g., [4], [5], [6], [7], [8], [9]), but as is demonstrated in [10] difficulties still endure. We follow along these lines in order to get closer to the two-abovementioned goals.

We argue that framework reuse problems are crucially related to a whitebox perspective, and to the lack of documentation about runtime structure. Hence, we turn to blackbox frameworks and component-oriented programming, to define a model for frameworks aimed at facilitating reuse. Furthermore, the level of support that tools can provide is decisively dependent on the kind of available framework documentation, which in turn is intimately tied to the considered framework model. We discuss these arguments more thoroughly in [12], and in this paper, we illustrate their relevance and use through the description of a case study application framework. We analyse how the model influences the structure of a framework for classification systems and how the documentation approach can be used to describe it. This case study illustrates how the proposed framework model and documentation approach alleviate difficulties of framework reuse, namely its comprehension and instantiation.

[1] This definition is more specific but agrees with that of IEEE 1471-2000 Standard or ISO/IEC 25961 Draft International Standard.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

We start, in section 2, by presenting proposed framework model. In section 3, we discuss the foundation for our documentation approach and describe the proposed notation. In section 4, we introduce the Classification System Framework. Section 5, illustrates the application of the framework model and proposed notation to the description of the case study framework. Related work is reviewed in section 6. Finally, in section 7, we present the conclusions and future work.

2. Framework model

Programming and design choices influence the way frameworks and, more generically, reusable software are built. They determine framework characteristics that are decisive to the problems and complexities that can arise when reusing frameworks, e.g. comprehend design rationale, identify variation-points, how to implement adaptations, understand how the framework can be composed with other software and difficulties in setting up the running application. Framework properties influence the requirements on documentation approaches and the kind of support that tools can provide. For example, blackbox frameworks enable the implementation of visual builders to help in its instantiation. On its turn, framework documentation depends on the characteristics of the framework to be described, e.g. small or large size, *whitebox* or *blackbox* [13], *called* or *calling* [14].

Part of the aforementioned problems are related to: whitebox reuse, which exposes details that hinder design rationale; inheritance that introduces implementation dependencies that complicate adaptation and can easily break the architecture; lack of clear framework interfaces for composition; and, obscure runtime structure that makes hard to instantiate the application. Our framework model addresses these issues, namely it approximates the framework concept to components, introduces the service-point concept, fosters interfaces and blackbox reuse, and exploits role modelling advantages, and applies component-oriented programming at the internal framework level. The UML class diagram of figure 1 depicts the resulting conceptual and structural framework model.

2.1 Frameworks and components

The proposed framework concept is inspired by the idea of framelets [15], and it is matched with the definition of components given in [16]. We consider frameworks to be small, not assuming full control of the application, and deployed as components. Contrary to framelets, they do not have an explicit limit size, because it is very dependant on the addressed problem or domain of application. Likewise, we propose frameworks that can be used independently and do not have to be integrated in a family of related frameworks. Perhaps a more influential characteristic of this model is that frameworks do not assume to be in full control. Thus, frameworks must provide means for its operation to be controlled externally, which is crucial for the problem of composition. We consider frameworks that, just like components, provide services and mechanisms to (partially) control those services (e.g., initiating a task). They are packaged and deployed to be reused by third parties as components, i.e., to be composed with other such frameworks to build complete systems. However, they retain most of the flexibility that frameworks traditionally provide.

We consider the framework to be constituted by *internal components*, which provide separate *computational* and *compositional interfaces*. An internal component is a class or a very small group of classes, represented by one of them. In the context of object-oriented languages, internal components are object-oriented components, i.e. they are not full-fledged components because they are not independently deployable. However, internal components are reused as is, in the context for which they were developed. Third parties compose their instances to configure the framework component. Our component-oriented approach does not consider, and is independent of, any standard component model and respective infrastructure.

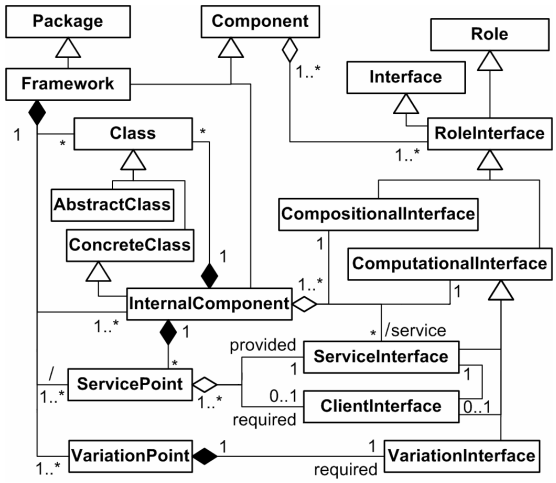


Figure 1. The framework model.

2.2 Service-points

Frameworks that communicate with application specific components solely through inversion of control (calling) do not provide services to control its execution. On the opposite side, frameworks that offer only services to clients (called, or class libraries) can-

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

not be tailored to application specific needs in a disciplined way. Consequently, we explicitly support and foster frameworks that have neither a purely called nor a purely calling architecture, in order to offer composability and adaptability, which are two crucial features for effective reuse.

To support this perspective, we introduce the concept of *service-points*: parts of the framework interface that are designed to be called by clients. The service-points are anticipated by framework developers and identify services that play a key role in the framework operation. The interaction between clients and framework can vary from single method invocation to complex protocols that impose obligations on the clients. Similar to what is practice in component-oriented designs, these obligations can be expressed by required interfaces. So, a service-point is comprised by a provided *service interface* and a corresponding required *client interface*. The framework is called by clients (implementing the required interface) through the service interface. The client interface is optional, because service-points do not always need to impose constraints on clients.

2.3 Interfaces and blackbox reuse

We propose a model in which frameworks variation-points are implemented exclusively by object composition with message forwarding via interfaces. This approach avoids many problems associated with inheritance. Programmers provide application specific components that implement the interfaces of variation-points. The framework calls the adapting components through these required interfaces, which we name *variation interfaces*. Inheritance remains as a useful mechanism to reuse implementation inside internal components.

In the literature, the term blackbox framework conventionally designates a framework that provides a selection of components to fill in all or most of its variation-points. According to this point of view, applications are developed exclusively by composing objects: instantiating the objects, setting their state and interconnecting them. As result, this kind of blackbox framework is hardly composable. Service-points alleviate this problem, but more adaptability is necessary. The way service-point clients compose frameworks – implementing required interfaces – is also the way application specific components adapt the variation-points. For this reason, our notion of blackbox framework includes the implementation of required interfaces uniformly, no matter if they belong to service-points or variation-points. Moreover, we relax our blackbox reuse concept to the point of ignoring whether framework designers or application developers are the ones that provide the components for (required interfaces of) variation-points. Although this is not the traditional understanding, it does not contradict the seminal definition given (by Johnson & Foote) in [13].

2.4 Exploiting role modelling

Role modelling can be considerably useful for the development of frameworks, as is illustrated in [17], because role models can be composed without being prematurely tied to a code structure that implements them. In what regards framework reuse, role modelling brings in several advantages, such as: a flexible and more natural way of defining restrictions upon clients; and, role interfaces are smaller than monolithic or "fat" object interfaces, which reduces coupling and provides more flexibility and reusability.

In [18] it is applied at design level, defining framework blackbox use by clients with explicit restrictions. Its articulation with class models for framework whitebox adaptation through inheritance is also defined. We extend its appliance to blackbox frameworks at implementation level, namely for adaptation through object composition via interfaces. This way role modelling flexibility and reusability advantages apply uniformly to the complete framework interface. Variation interfaces describe roles to be played by application specific objects that are invoked by the framework. Client interfaces describe roles to be played by client objects, defining constraints for using the framework services. And, service interfaces describe roles played by framework objects that identify framework services to be called by clients.

3. Framework documentation

The model previously described shapes the documentation constructs described in this section. We propose a formal prescriptive documentation approach that can be used by tools in order to provide suitable support to application developers for instantiating frameworks. First, we discuss the documentation approach, and then, we address the proposed constructs.

3.1. Notation format

Complete framework documentation covers several aspects of framework description (application domain and objective, static structure, dynamic behaviour, how to use it, example applications, *etc.*), which can address various kinds of reuse scenarios (instantiation, maintenance, design mining, *etc*) with different needs. From this diversity, we are interested in documentation that can help to describe frameworks and, at the same time,

can be handled by tools providing assistance in the instantiation activity.

Although describing the design is a framework developer responsibility, the chosen documentation approach is decisive to the following phases of the reuse process to be carried out by application developers. This issue has two sides: the reuse possibilities, and the amount of support (or automation) that can be provided by tools. In [11] we discuss both sides of the question and we conclude that descriptive techniques are necessary to support unanticipated reuse (with opened possibilities), and formal notations are required to provide tool support.

In our opinion, to provide documentation that enables unanticipated reuse is vital for the success of frameworks as a reuse paradigm. Hence, we take a descriptive approach, but we augment the reuse flexibility it offers with as much guidance as possible. Namely, we propose a descriptive technique supporting the explicit description of variation-points and service-points.

Since we look for documentation that can be processed by tools, we follow a formal documentation approach. However, the mathematical notation of most formal languages can be an impediment to its wide use by application developers. Additionally, visual notations have decisive advantages over textual languages, when it comes to communicate software design and architecture. Unsurprisingly, UML comes up as a natural choice to describe frameworks, because it is the standard object-oriented design notation in industry. Furthermore, it is a general-purpose notation that can be extended by domain specific profiles.

3.2. The UML-FD profile

Generic object-oriented notations (e.g. UML) provide constructs for the design of applications, but they lack support for concepts specific to framework reuse. The only such concept that UML 2.0 [19] provides is the «framework» stereotype, applicable to package metaclass in order to identify packages that modularize a framework. One of the important aspects that have been widely acknowledged is the need to explicitly identify framework variation-points (e.g., [4] [5] [8] [9]), and define them. Whether a class or an interface is part of a variation-point or not, cannot be derived from its intrinsic properties, neither can it be derived from the domain modelled by the framework. Likewise, whether or not an interface is part of a service-point, and whether it is a client interface or a service interface, has to be (explicitly) specified. Both variation-points and service-points are always conscious design decisions made by framework designers; altogether, they describe the framework interface or API. Thus, they are fundamental concepts that must be clearly identified and described, in order to provide guidance for reuse activities. They should be first-class entities of framework modelling.

The UML Profile for Framework Description (UML-FD) extends UML with annotations dedicated to support framework concepts. It provides constructs for variation-point and service-point explicit specification, and other relevant structural elements, according to the previously defined framework model. The UML-FD profile is defined for UML 2.0, making use of its improved extensibility mechanism. UML 2.0 profiles are consistent with the more general Meta Object Facility (MOF) extension mechanism, being defined as a specific meta-modelling technique: stereotypes are specific metaclasses, tagged values are standard metaattributes, and profiles are specific kinds of packages.

Figure 2 defines the UML-FD profile abstract syntax and its integration with the UML 2.0 meta-model. Below, we describe the UML-FD annotations for the description of frameworks. All the UML meta-classes extended by UML-FD belong to the Classes language unit. According to the nature of the extensions and its purpose, they are arranged in three groups: component structure, blackbox variation-points, and service-points.

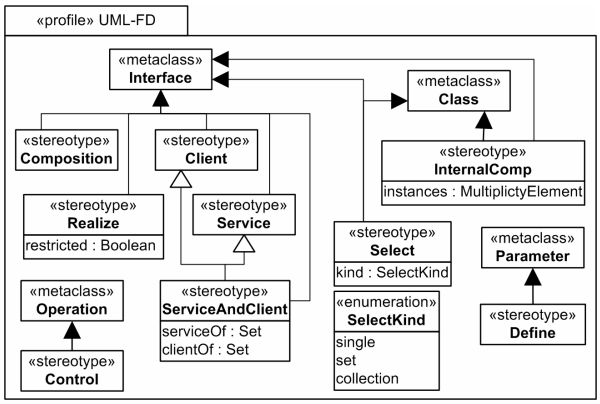


Figure 2. The UML-FD profile for UML 2.0.

3.2.1. Component structure. The «InternalComp» stereotype can be applied to framework classes and interfaces. It identifies a framework *internal component*. It has a property named instances that limits the number of instances that can be created for the respective component, which is useful for example to document singletons ('1..1') and, more broadly, to express probable scalability boundaries of the architecture. The class/interface gives the full type of the component, and the component instances encapsulate any "internal" objects.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

The «Composition» stereotype can be applied to an interface of an internal component. It identifies a *composition interface*, which defines the methods (whose names typically start with 'set', 'add', 'remove', *etc.*) necessary to configure the component instances and link them with other instances.

3.2.2. Variation-points. The «Realize» stereotype is applicable to interfaces required by framework components. It identifies an opened *variation interface* that defines an abstract type for which application specific components can/must be provided. It has a property named restricted that if true forbids subtyping (implementations cannot have a more specific interface).
The «Select» stereotype is applicable to interfaces and abstract classes (for increased flexibility in framework implementation), for which the framework provides a collection of implementations. It identifies a closed *variation interface* that limits the specialization options to those stipulated by framework designers. It has a property named kind, whose value can be single (only one kind of components allowed), set (several kinds of components but no more than one instance of each kind), or collection (several components possibly of the same kind).
The «Define» stereotype is applicable to parameters of operations in component interfaces. It identifies a parameterized variability that defines an important characteristic of the framework (e.g., in opposition to ordinary parameters for component interconnection).

3.2.3. Service-points. The «Service» stereotype can be applied to interfaces provided by framework components. It identifies a *service interface* that application specific components can use to interact with the framework. The *service interface* can have one corresponding *client interface* that is identified by an association between the latter and the internal component realizing the former. This construction describes the most powerful form of *service-point*.
The «Client» stereotype can be applied to interfaces required by framework components. It identifies a *client interface* to be implemented by application specific clients that use the corresponding *service interface*. Any (callback) methods it contains define client constraints, namely static behaviour obligations.
Note that different framework components define different service-points that can involve the same service and/or client interfaces. Moreover, the same interface can have opposite semantics in more than one service-point. To handle this problem, the «ServiceAndClient» stereotype can be applied to interfaces that are both provided and required by different framework components. It includes the properties serviceOf and clientOf that contain a list of the components' names for which the annotated interface has the service and client semantics, respectively. Tools can filter the display of the semantics applicable to each service-point by looking up the components' names in the lists.
The «Control» stereotype can be applied to operations of framework internal components that play a key role in the framework operation. It identifies a *control operation* to be called by clients in order to externally control a special framework function or trigger an important event. This is the minimal form of *service point*, without a separate service interface.

It is worth to mention that the newly introduced metaclasses «Composition» interface, «Select» interface and «Define» parameter, only come to play during the application instantiation phase. They reflect the dynamic needs of the run-time component composition. The composition interface is used to interconnect components, and the other two are blackbox variation-points that are defined (concretized) during the application definition.

4. The Classification System Framework

Classification systems are used for quality control, by classifying products according to the level of compliance with predefined or standard properties. They read a number of sensors monitoring products properties and, in consequence, control actuators that make the classification. Typically, there is a trigger sensor that signals a product arriving at the system. This is the start of a classification cycle: data is collected by the sensors, then it is captured in a centralized representation and processed until it can be compared with reference values; according to the comparison the product is categorized (e.g., separated or tagged). These are the commonalities typical of this kind of systems. The required variabilities include the properties to measure, the kind of measurement transformations, and the actions for each possible classification.

We have developed an object-oriented Classification System Framework (CSF) that borrows its design from [20] introducing small changes (see figure 3). There are three MeasurementEntitys: Sensor, MeasurementValue and ClassificationItem. Sensors measure particular variables of the item being classified. There are ConcreteSensors that interface real-world sensors in a one-to-one relationship, and AbstractSensors that compose ConcreteSensors (e.g., for calculating the average of the values of two sensors reading the same property). The three sensor classes implement an instance of the *Composite* design pattern [21]. Another instance of the same design pattern is found at the MeasurementValue hierarchy. A MeasurementValue represents a property of the item to be categorized, such as weight,

size or temperature. It is responsible for the comparison between the value measured at the current item, and the reference value stored during system calibration. A SimpleMeasurementValue represents a property of the item directly obtained from a Sensor, which are generally zero-dimensional values. CompoundMeasurementValue represent properties that are composed of SimpleMeasurementValues; for example, it may contain several values from the same sensor obtained at different instants (one-dimensional), values from different sensors that are integrated into another value, or multi-dimensional data from one sensor. The ClassificationItem defines the top-level behaviour for the classification cycle, namely to collect the data from the physical item into MeasurementValues, and command Actuators to classify the item according to the comparison with reference values. The Trigger is a subclass of ConcreteSensor that is responsible for notifying the ItemFactory when an item enters the system. The ItemFactory is an instance of the *Abstract Factory* design pattern [21] and it is in charge of creating ClassificationItems, configure their Actuation and activate them.

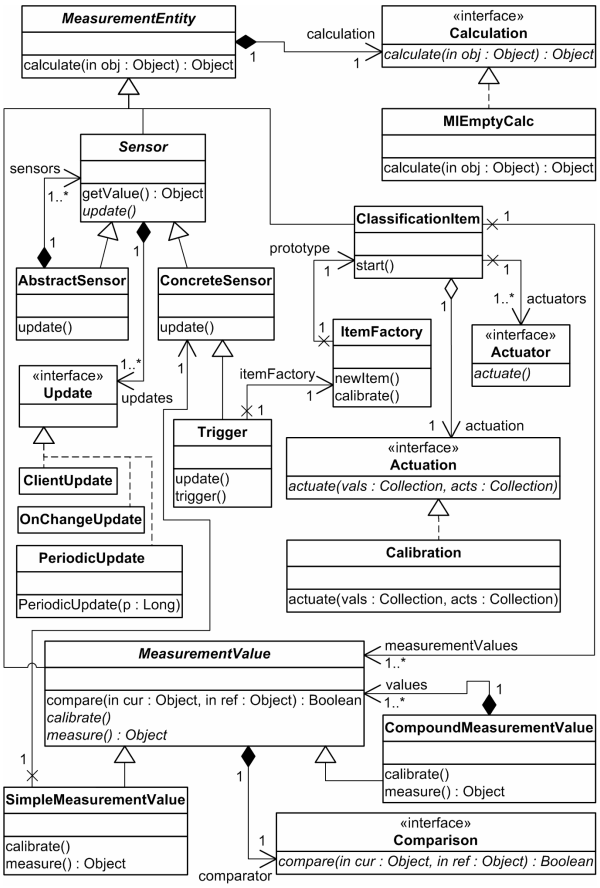


Figure 3. The Classification System Framework.

The variabilities of the three MeasurementEntitys are implemented by instances of the *Strategy* design pattern. The way Sensors are updated depends on the application and kind of sensor. The update of Sensor's value is abstracted by the Update interface and three cases have been identified: ClientUpdate (when the value is requested), PeriodicUpdate (when a period starts) and OnChangeUpdate (when the physical sensor notifies Sensor either via hardware interrupt or software event). The meaning of the disparity between reference values and measured values in MeasurementValues is also application specific. For this reason, we introduce the Comparison interface to the original design. The Actuation interface invoked by ClassificationItem enables the customization of the classification process, i.e. it determines which actuators to act upon and which actions to perform on the physical item. The Calibration class defines a specific kind of Actuation strategy that, instead of commanding the actuators, it rather instructs the MeasurementValues to calibrate themselves. Finally, there is a variability covering a wide spectrum of needs that is implemented through the Calculation interface. Each MeasurementEntity needs to make calculations to obtain its value(s): Sensors have to communicate with physical sensors, MeasurementsValues need to process sensor's data, and ClassificationItems have to collect data from sensors. In the last case, a specific Calculation has been identified – CIEmptyCalc – that simply forwards the measurement message to MeasurementsValues. A detailed description of CSF dynamic behaviour is not relevant for our purposes.

As a final note, it is important to highlight that frameworks like the CSF cannot reach the level of traditional blackbox frameworks, i.e. even when they reach maturity they cannot provide the components necessary to fill in its variation-points. This happens simply because it is not possible to anticipate, for example, the calculations for sensors that, inevitably, are application specific. Imminently, this is imperative for embedded systems, which always have to interface hardware.

5. Applying the framework model and UML-FD annotations

Being an example of good design, the CSF framework already adopts parts of our proposed model, namely: it is small, uses object composition via interfaces, and its variation-point required interfaces are role interfaces. Even though this reduces a small part of the difficulties described in section 2, it still takes considerable effort to instantiate the framework, because there is neither explicit identification nor charac-

terization of the variation-points, the service-points are not described, and there is no internal component structure. The UML-FD allows the description of these aspects of the framework, as figure 4 shows.

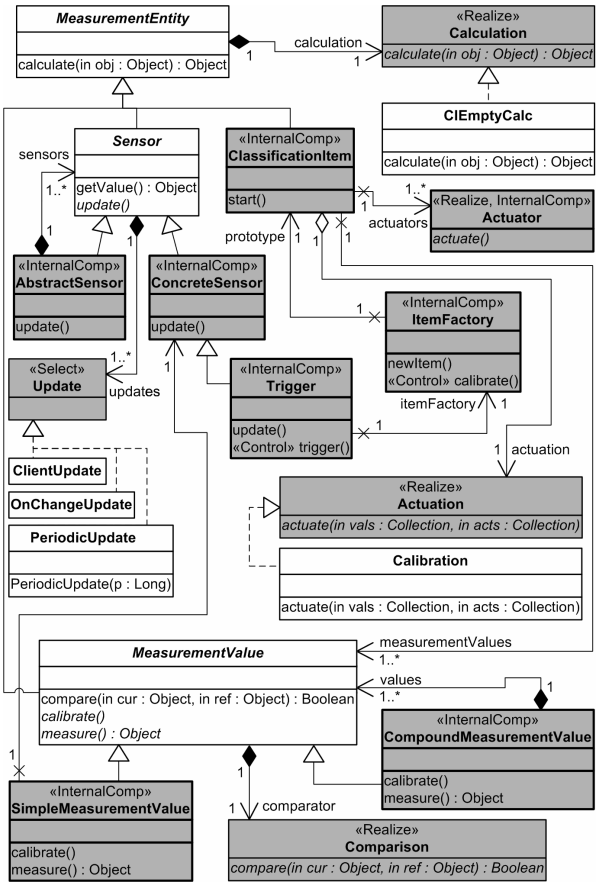


Figure 4. The CSF described by UML-FD.

5.1. Blackbox variation-points

As described in the previous section, the CSF has five variation-points: (MeasurementEntity's) Calculation, (Sensor's) Update, (MeasurementValue's) Comparison, (ClassificationItem's) Actuation, and Actuator. All of them are implemented by object composition via interfaces. Therefore, most of them are annotated with the «Realize» stereotype, indicating that it is necessary to provide specific application implementations for them. The exception is Update, which gets a «Select» annotation indicating that it is a blackbox variation-point in the traditional sense (i.e. a *closed* variability). The allowed values are ClientUpdate, PeriodicUpdate and OnChangeUpdate and, since these are not mutually exclusive, the kind property of «Select» has the value set.

Although the «Define» annotation is not used in this example framework, it can be illustrated from the Update variation-point. Both ClientUpdate and OnChangeUpdate are simply 'tag' classes, which do not exhibit any behaviour. If this was also the case for the PeriodicUpdate class, all of them could be represented by enumerated values. This way the Update variation-point could be established as a parameter of class Sensor of type Enumeration, embedding the handling of the three cases exclusively inside Sensor. In this kind of scenario, the «Define» stereotype can be applied to the parameter of the method responsible for closing the variation, indicating that it is a variation-point.

MeasurementValue's calculation and comparison algorithms are implemented as two separate role interfaces, to be realized by components linked to MeasurementValue by object composition. However, both algorithms might have been combined in a single more specific interface, forcing their realization by a single component, or, even worse, as abstract methods, forcing their implementation by deriving MeasurementValue. Both of these hypothetical cases increase the coupling between MeasurementValue and the variation algorithms' implementations, and reduce the flexibility of the objects' composition. With role modelling the coupling is kept to a minimum (by role interfaces) and the realization structure can vary (from one component for both, to an independent component for each), greatly increasing the reusability.

5.2. Component structure

The internal components have an intermediate granularity level between objects and full-featured components. It is easier to compose few large components than many small components, but the flexibility also decreases, as pointed in [16]. Moreover, configuring components that include too many objects can become cumbersome, because it may include too many parameters for application developers to set. This can make the interface of tools supporting component composition become less intuitive. Besides, it also imposes higher demand on the tools ability to interpret the elements of the design of each internal component. Therefore, the internal components should have a granularity that balances the flexibility of composition with the ease of configuration.

Framework designers using UML-FD should take into account the above considerations when defining the framework structure. Our approach considers internal components to be those classes whose instances compose other objects that contain a minimum of configuration parameters (ideally zero). We consider as internal components seven concrete classes and an interface, respectively: Trigger, ItemFactory, ClassificationItem, ConcreteSensor, AsbtractSensor, SimpleMeasurementValue, CompoundMeasurementValue and Actuator.

All of them are then annotated with the «InternalComp» stereotype, however they have distinct values for the cardinality property. Trigger, ItemFactory and ClassificationItem have the value '1..1', which indicates that an instance of each must be included in the final application. The remaining ones have the value '1..*', which indicates that at least one instance is required but there is no explicit limit (naturally, the multiplicities of the associations in the static structure diagram must be observed).

From the perspective of framework communication, too many annotations can make the design diagrams difficult to read. This could be alleviated by placing the annotation in abstract superclasses, instead of concrete subclasses, and defining that by convention any concrete subclass should be treated as an internal component. However, this approach has disadvantages, because the components could only be handled using the same interface of the annotated superclass.

In relation to the composition interfaces, the «Composition» stereotype is applied to one interface of each internal component (this is not represented due to space restrictions and figure readability). In terms of UML diagrams, its representation can vary from one interface realized by each component to a separate compartment in each internal component class. We prefer the latter approach because it keeps the diagrams more clear. However, this option requires tools to implement that kind of representation for class methods that originate from «Composition» interfaces.

5.3. Service-points

Although a framework can be integrated with other frameworks (or with a platform) through variation-points, service-points constitute the dedicated interface for framework composition. The CSF has to be integrated with a platform, or some hardware interface layer, to make the bridge with hardware sensors and actuators. Additionally, it can be composed with a graphical user-interface framework, or a simpler user-interface that at least inputs the calibration command.

The reading of hardware sensors is done through the calculation() method of Sensor, while the output to hardware actuators is accomplished by the actuate() method of Actuator. However, the framework has additional interaction points, namely to receive events for updating OnChangeUpdate sensors, triggering the Trigger and ordering the calibration of ClassificationItem. These three interaction points need to be clearly identified and, therefore, they are described as service-points (the first two in figure 5, and the last one in figure 4).

The service-point for updating Sensor components with an Update of kind OnChangeUpdate includes: the «Client» interface HWSensor describing the role of a client entity (hardware interrupt and/or software event mechanisms) with the obligation to use the service; the association between the client interface and Sensor, with an association end named sensorUpdate, that identifies the respective service interface; and, the «Service» interface SensorUpdate describing the update() method as the service to be invoked. A similar construction holds for triggering the Trigger and calibrating the ClassificationItem. The difference is that the involved framework services are *control operations*, respectively the trigger() method of Trigger and the calibrate() method of ItemFactory. Another relevant different is that the calibration() control operation imposes no constraints on the client whatsoever, and thus it has no associated client interface.

The SensorUpdate and trigger() service-points have empty client interfaces, because they identify roles with dynamic obligations only (invoke the identified services). Observe that both associations are not navigable from the framework side, which means the framework does not depend on them. Therefore, since they do not constitute required interfaces, they exist only at the description level. These examples show how naturally the role modelling technique identifies required clients and enables the definition of the respective obligations (dynamic only in these cases).

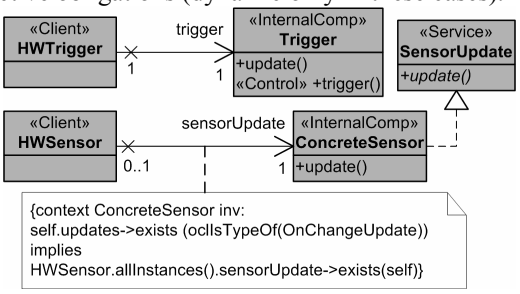


Figure 5. Service-points with UML-FD

6. Related Work

The framework reuse problem has been addressed from prescriptive informal textual language approaches (e.g. [7]) to descriptive formally defined visual notations that extend UML [4], as well as mixed prescriptive-descriptive (e.g. [10]). Previous descriptive documentation approaches [2], [4], [5], [6], [8], [9], [10], [11], [22], were proposed independently of a framework model. Although not explicitly stated they implicitly assume a few characteristics, generally large-sized, whitebox, calling frameworks.

A framework model that, similarly to ours, also incorporates role modelling concepts and components is described in [23]. It considers that variation-points can be role interfaces, abstract classes and component implementations, depending on the needs of the applica-

tion to be built. The model leaves aside how application specific components relate to these three kinds of variation-points, and whether variation-points are predefined (specified) or not. It considers a wide range of framework adaptation mechanisms, including a whitebox perspective using inheritance. It assumes more the perspective of a model for design and development of frameworks for in-house reuse (or opened framework reuse), than the perspective of a model for designing and structuring frameworks for third party reuse.

Specialization patterns [6] describe the static dependencies between structural elements of object-oriented programs. They are implementation oriented (too detailed) and cover both the framework adaptation and the application instantiation. Consequently, they define an application structure, prescribing the reuse options. Their representation is defined by a dedicated graph notation, which lacks visual tool support. Instead, they are defined as a textual tree-like structure, in a tool supporting application development at the source code level.

Catalysis [17] also extends UML to support reuse. However, it defines model frameworks as collaborations of abstract types, which are reused through parameter substitution. It does not address the reuse of (code) frameworks, and consequently it does not provide dedicated annotations for explicit representation of its variation-points. Catalysis defines a software development method based on the concepts of model frameworks and components.

A few works have been devoted to descriptive visual approaches for documenting frameworks. UML has been the obvious target, being extended with concepts dedicated to framework documentation [5], [4], [8]. These works have similarities – all provide UML extensions to identify variation-points – and parts that are complementary: [5] focus more on variation-points identification and characterization, while [4] provides stronger support for expressing framework syntax and semantics, and [8] introduces selection of black-box components and parameterization. However, they consider mostly whitebox reuse and none of them supports called frameworks. Application developer adaptations are only possible through inheritance relationships; object composition via interfaces is overlooked as a mechanism to support blackbox opened variation-points. Usage relationships with restrictions on clients are neglected.

The role modelling technique [18] is a complementary technique that tackles object relationships, which are fundamental for framework integration and composition. The requirements on clients calling framework services are explicitly represented but, on the other hand, variation-points are considered only through inheritance (whitebox framework adaptation). Moreover, variation-points are implicitly identified by abstract classes, and therefore, it disregards the explicit identification of variation-points.

7. Conclusion and Future Work

We propose a novel model for framework design, and a UML based notation for framework description. The model extends, or is inspired by, a few known techniques (components, role modelling, and framelets), and introduces a new concept (service-points). The UML-FD notation is based on the model and, comparatively to previous proposals, it provides a fundamental variety of blackbox variation-points and a wider coverage of the different reuse needs. Both the model and the notation, individually or synergistically together, contribute to alleviate the difficulties that application developers face when reusing frameworks. We discuss and illustrate their use with the Classification System Framework.

There is no need to artificially downgrade the design of the CSF to demonstrate the advantages of our proposed model and notation. Small frameworks improve reusability and composability. Object composition via interfaces and internal components hide implementation dependencies, which facilitates adaptation and minimizes the risk of breaking the architecture. Role interfaces provide more reuse flexibility for both adaptation and composition, and enable the definition of constraints for framework clients. Our concept of blackbox framework combines the composability through called service-points with the adaptability of calling variation-points. Explicit identification and characterization of variation-points and service-points help application developers to learn how to adapt and compose the framework, respectively. Internal components provide information about the runtime structure of the architecture, which is vital for understanding how to instantiate the final application. Finally, these aspects are synergetic and all contribute to a better comprehension of the framework design and architecture.

Object composition via interfaces and internal components allow the decomposition of the framework instantiation process in two different reuse activities: the development of application specific components (*framework adaptation*), and the composition of components' instances for building the executing application (*application instantiation*). This way the application developer assignment is divided in two smaller steps. In addition, both activities can be more systematic due to the separation of computational and compositional interfaces.

The UML-FD constructs can be handled by tools. Future research includes the development of generic

tools, that accept as input UML-FD framework models, to provide support for both framework instantiation activities. For framework adaptation, a tool can interpret the annotations of variation-points and service-points, and provide guidance for the implementation of application specific components (which is an inherently creative task). The result is an extended framework model. For application instantiation, a tool (usually known as *visual builder*) can interpret the annotations of internal components and composition interfaces, and support graphical composition of components. The final application is defined by diagrams - representing components' instances, their state configuration and interconnections - from which the application code is automatically generated.

8. References

[1] D. Parsons, A. Rashid, A. Speck, and A. Telea, "A framework for object-oriented frameworks design", in *Proc. of TOOLS'99*, IEEE Society, 1999, pp. 141-151.

[2] W. Pree, *Design Patterns for Object-Oriented Software Development*, Addison-Wesley, Reading, MA, 1995.

[3] J. Bosch, P. Molin, M. Mattsson, P.O. Bengtsson and M. Fayad, "Object-oriented frameworks – problems & experiences", in *Building Application Frameworks*, M. E. Fayad, D. C. Schmidt and R. E. Johnson, Ed. New York, NY: Wiley & Sons, 1999, pp. 55-82.

[4] N. Bouassida, H. Ben-Abdallah, F. Gargouri and A. Ben Hamadou, "Formalizing the framework design language F-UML," in *Proc. of the 1st IEEE International Conference on Software Engineering Formal Methods*, 2003, pp. 164–172.

[5] M. Fontoura, W. Pree and B. Rumpe, *The UML Profile for Framework Architecture*, Addison-Wesley, 2002.

[6] M. Hakala, J. Hautamäki, K. Koskimies, J. Paakki, A. Viljamaa and J. Viljamaa, "Annotating reusable software architectures with Specialization Patterns," in *Proc. of the Working IEEE/IFIP Conference on Software Architecture*, August 2001.

[7] L. Murray, D. Carrington and P. Strooper, "An approach to specifying software frameworks," in *Proc. of the 27th Conference on Australasian Computer Science*, Dunedin, New Zealand, 2004, pp. 185–192.

[8] T. Oliveira, P. Alencar and D. Cowan, "Towards a declarative approach to framework instantiation," in *Proc. of the Workshop on Declarative Metaprogramming to Support Software Development of the 17th IEEE ASE*, Edinburgh, Sept. 2002, pp. 5-8.

[9] A. Ortigosa, and M. Campo, "SmartBooks: A Step Beyond Active-Cookbooks to Aid in Framework Instantiation", in *Technology of Object-Oriented Languages and Systems*, IEEE Press, ISBN 0-7695-0275-X, June 1999.

[10] D. Kirk, M. Roper and M. Wood, "Identifying and Addressing Problems in Framework Reuse", in *Proc. of the 13th International Workshop on Program Comprehension*, 2005.

[11] S. Lopes, A. Tavares, J. Monteiro, and C. Silva, "Describing Framework Static Structure: promoting interfaces with UML annotations", In *Proc. of the 11th International Workshop on Component-Oriented Programming* of the *20th ECOOP*, 2006.

[12] S. Lopes, A. Tavares, J. Monteiro, and C. Silva, "An Architectural Model and Description Notation for Composable, Adaptable, Blackbox Frameworks", *submitted to publication*, 2007.

[13] R. Johnson and B. Foote, "Designing reusable classes," *Journal of Object-Oriented Programming*, vol. 1, no. 2, Jun. 1988, pp. 22-35.

[14] S. Sparks, K. Benner and C. Faris, "Managing object-oriented framework reuse," *IEEE Computer*, Sep. 1996, pp. 53–61.

[15] W. Pree and K. Koskimies, "Framelets – small and loosely coupled frameworks", *ACM Computing Surveys*, vol. 32, nº 1, ACM Press, New York, NY, USA, 2000.

[16] C. Szyperski, *Component Software – Beyond Object-Oriented Programming*, 2nd Edition, Addison-Wesley/ACM Press, 2002.

[17] D. D'Souza and A. Wills, *Objects, Components and Frameworks With UML: The Catalysis Approach*, Addison-Wesley, 1998.

[18] D. Riehle, and T. Gross, "Role model based framework design and integration," in *Proc. of the 13th ACM Conference on Object-Oriented Programming Systems, Languages and Applications*, Vancouver, Canada, 1998, pp. 117–133.

[19] Object Management Group, *Unified Modeling Language: Superstructure, version 2.0*, Object Management Group, August 2005.

[20] J. Bosch, "Measurement systems framework", in *Domain-specific Application Frameworks*, M. E. Fayad, D. C. Schmidt and R. E. Johnson, Ed. New York, NY: Wiley & Sons, 2000, pp. 177-205.

[21] E. Gamma, R. Helm, R. Johnson and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*, New York, USA: Addison-Wesley, 1995.

[22] P. Steyaert, C. Lucas, K. Mens and T. D'Hondt, "Reuse Contracts: Managing the Evolution of Reusable Assets", in *Proc. of the 11th Conference on Object-Oriented Programming Systems, Languages and Applications*, San Jose – CA, USA, ACM Press, October 1996, pp. 268-285.

[23] J. Gurp, and J. Bosch, "Design, implementation and evolution of object oriented frameworks: concepts and guidelines", *Software Practice & Experience*, 31(3), 2001, pp. 277-300.

Patterns for Integrating and Exploiting Some Non-Functional Properties in Hierarchical Software Components

Hervé Chang and Philippe Collet
Université de Nice Sophia Antipolis
Laboratoire I3S, CNRS UMR 6070
2000, route des lucioles Les Algorithmes - bât. Euclide B, BP 121
06903 Sophia Antipolis France
{Herve.Chang, Philippe.Collet}@unice.fr

Abstract

Providing powerful and fine-grained capabilities for the analysis and management of non-functional properties is a major challenge for component-based software systems. In this paper, we propose integration patterns for non-functional properties of hierarchical software components. These patterns are based on a classification of low-level non-functional properties, which takes into account their nature and lifecycle. They make explicit the implementation of these properties in relation with components and can be used to develop some forms of compositional reasoning. The proposals are exploited in non-functional contract negotiation by enabling a negotiation process to be precisely propagated down the component hierarchy.

1 Introduction

In component-based software engineering, one of the major challenges is still to facilitate the management of non-functional properties. These properties represent various qualities of software components and systems, such as their runtime qualities or system lifecycle. With the need to manage more complex and long-running component-based systems, identifying and handling these properties as precisely as possible during design, (re-)configurations and runtime phases is crucial. Moreover, in order to reason on the composition of non-functional properties, support to determine properties of component assemblies from properties of individual components must also be provided.

Over the past decades, numerous studies have been conducted around the modeling, analysis and management of non-functional properties. At the highest-level of analysis, some approaches provide methodologies to analyze quality attributes [1], or address non-functional properties through quality standards (IEEE 1061, ISO/IEC-9126) and models [2, 4] that provide classifications of high-level properties. They aim at proposing a generic taxonomy and studying relationships between properties, or structuring knowledge by successively defining and decomposing non-functional characteristics. As these properties mostly remain at a high-level, they are not clearly modeled in relation with runtime components and platforms, and their specification and management is thus limited. On the other hand, at the lowest-level of resource management, substantial works exist on providing resource management and monitoring capabilities to applications, at different level of abstractions (API, technologies) [9]. Such works also aim at integrating advanced tools for the diagnosis of performance issues [18]. Besides, to achieve non-functional requirements, numerous works in the domain of distributed systems propose some component-based middleware platforms that provide QoS control and measurement capabilities through reflective and adaptive systems techniques [12, 16]. They particularly focus on critical network-related properties, and provide integrated control mechanisms without explicit representation of non-functional properties. Some other component platforms also enable flexible integration of arbitrary non-functional services using containers. Containers wrap set of components, and are responsible for handling essentially non-functional middleware-related services (transactions, load balancing, security checks, etc.). They are integrated using code transformation such as aspect-weaving, or indirection frameworks like interceptors and meta-object protocols.

Several compositional approaches, which are specific to component-based systems, aim at improving non-functional property analysis in component assemblies. Analysis models and property theories are thus integrated to component technology [13], and they allow one to guarantee, by construction, the predictability of some properties on compo-

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

nent assemblies. However, they require advanced analysis models and techniques, and are mostly dedicated to specific properties, such as latency [13], reliability [19] or memory usage [11]. Some of these models could be extended to other properties if the properties are properly related to the architecture and modeled in some generic ways that make possible to reason on them. Recently, to make reasoning on non-functional properties possible at the architectural level, the relationship between software components and software architecture has been outlined [21] and exploited by studying how properties relate to component assemblies and individual component properties. In particular, in order to help describing how properties relate to compositions, an interesting classification [8] have synthesized different class of dependency between properties, components and their context.

Our work stands half-way between analysis techniques and management systems. We propose to reify non-functional properties in relation with components, and to provide means to support a basic form of compositional reasoning on these properties. This should then enable software architects to better master the modeling, integration and also the runtime management of non-functional properties into component-based systems. In this paper, we thus present some simple architectural patterns that model non-functional properties. They are based on a classification of some low-level observable non-functional properties, which is established by considering their nature and lifecycle. The proposed patterns reify different kinds of parameters on individual components, as well as physical resources. As for compositional properties, a reasoning support based on meta-level elements is also described.

The rest of the paper is organized as follows. The next section describes the proposed classification and patterns at an abstract level, as well as the support for compositional properties. The mapping of these patterns to the hierarchical component platform *Fractal* [5] is described in section 3. Section 4 illustrates the use of the proposed patterns for contract negotiation in *Fractal*, especially to propagate negotiations down into the component hierarchy. Section 5 concludes this paper and discusses future work.

2 Classification and Modeling of Non-Functional Properties

2.1 Rationale

The proposed classification is not intended to be exhaustive. It is rather limited to the range of low-level properties which are measurable and sufficiently orthogonal to functional aspects. Hence, non-functional aspects which concern high-level properties and system lifecycle at development and maintenance phases, are not taken into account here, as well as other temporal aspects, which require more knowledge about the behavior of components. Morever, to reason compositionally on non-functional properties, the analysis of non-functional properties of a system must also be based on properties of the components that compose it. Consequently, the classification also takes into account the composability of properties at the level of component compositions. The classification is then built by first analyzing what kinds of non-functional properties can be directly derived from individual components, and what kind of features do they express in relation with them. The lifecycle of properties is also analyzed in relation with the one of components. The moments when these properties are defined are distinguished, as well as when they are to be observed.

The proposed patterns directly match the categories of non-functional properties at an abstract level of architecture specification in order to remain independent from underlying component technologies. They are also used to support automated reasoning on compositional properties, once and for all, at the level of patterns themselves.

2.2 Patterns

We first distinguish three categories of non-functional properties that directly match the concept of component attributes.

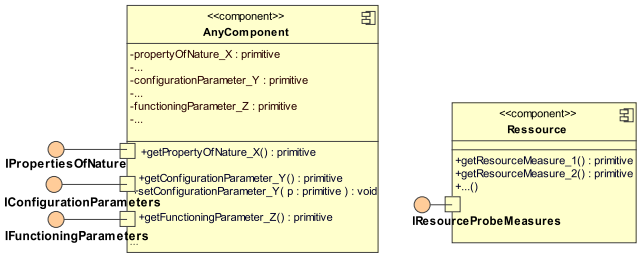
Some non-functional properties represent the key features or *nature of components*. For example, they can describe a memory footprint, the compatibility version number of a video codec or a maximum capacity. These properties are comparable to the technical characteristics of electronic or mechanic components, they capture and represent some design and development features of components and have an impact on their whole usage. They result of choices made when designing and developing components, and are generally taken into account at assembly and configuration times to evaluate the suitability of components, for instance when selecting potential components and matching them to requirements. As they describe the nature of components, they cannot be changed and their measurements remain constant all along configuration and runtime phases. The associated pattern models properties of this first category as read-only private attributes of components, which are accessed only by their associated getter operations defined in a provided interface. This is illustrated by the provided interface named `IPropertiesOfNature` on the left side of the two *UML* 2 component diagrams[1](see figure 1a).

Some other properties represent *configurable parameters* of components themselves. They describe, for exam-

[1] *UML* 2 components represent independent, interchangeable parts of a system. They realize one or more provided and required interfaces, which determine the behavior of components. Interfaces define sets of operations that components implement. Attributes can also be added to components to represent data fields or properties about them.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

ple, the size of a buffer component, the maximum size of a resource pool component. They influence the required and provided services of components, and may be (re-)parameterized to set components to some specific functioning mode, and adapt them to their runtime environment (other components, runtime infrastructure). They can be defined during the development phase by default values, but they are mostly observed and changed at assembly and (re-)configuration times, to properly customize the functioning mode of components. Their pattern models these properties as both read and write component attributes, with associated getters and setters defined in a provided interface (named `IConfigurationParameters` in figure 1a).



(a) (b)

Figure 1: Overview of patterns.

Others properties describe *functioning parameters* of components. For example, the number of active sessions on a web server, the current state of a video player processing a media stream, and the current number of packets exchanged between a given client and a server can all be considered as such properties. They capture some key information about the behavior of components at runtime and can thus be seen as properties that probe for some functional aspects of components. They are also modeled as read-only component attributes with their associated getter operations defined in a provided interface (named `IFunctioningParameters` in figure 1a).

The two other categories considered in our classification consist in properties that are related to the runtime infrastructure. These properties can represent physical resources such as memory, CPU as well as other network properties (bandwidth). They are generally considered under the general term of *resources*, and they describe exogenous requirements of applications. They clearly determine the execution of services, in term of external resources being provided by the underlying infrastructure. In order to take advantage of the component-based approach and to make possible their effective monitoring and management, it is now common to reify these resources as components at the application level. As a result, the pattern for resources corresponds to their reification as full-fledged components (figure 1b). Beyond the existence of resources, *resource capacity properties* are frequently considered, as they describe some aspects in relation with the resource use, such as memory occupation, battery utilization, CPU usage, or bandwidth level. These properties express and quantify the level of resources provided by the underlying infrastructure, and required by applications. The proposed pattern models them as operations of the reified resource components, and are accessed at runtime through a provided interface (named `IResourceProbeMeasures` in figure 1b).

As components may exhibit properties that belong to each of the categories, the proposed patterns can be applied together. Moreover, as they are only built upon standard elements of the supporting components, these patterns can be applicable to model non-functional properties on both primitive and composite components.

Regarding the measurement of non-functional properties, the proposed patterns provide standardized way to represent properties at the modeling level, but they do not provide predefined mechanisms to measure them, at the implementation level. Such mechanisms are defined at implementation time when the component technology and the runtime environment are determined. However, some non-functional properties that monitor infrastructure may still exploit these patterns, as they make explicit how non-functional properties of components are exposed and how properties may vary along the component lifecycle. Besides, properties implemented through read-only attributes, such as the general notion of reliability, could be statically computed by statistical measures on a system and then set at configuration times in the appropriate attribute. On the other hand, runtime properties, mainly related to resources, can be simply measured through apppropriate resource probes and asynchronous communications support for processing monitoring information, like the ones provided in the DREAM framework [14].

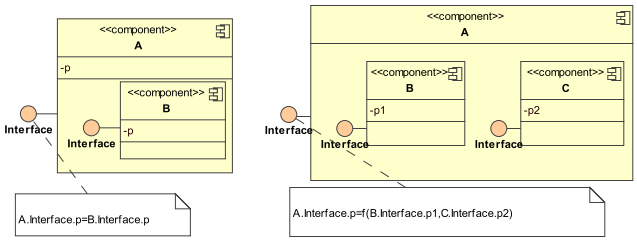
2.3 Enabling Reasoning on Compositional Properties

To be able to reason on the realization of a quantifiable compositional property from other ones, information describing the relationships, as well as some appropriate reasoning support must be provided.

Except resource properties, which do not express quantifiable properties, other properties from our classification have been modeled using patterns that derive directly from individual components. These properties are thus compositional by nature, and some simple form of compositional reasoning can be supported. Building on those primitive parts, we define a *compositional property* as a property providing the following characteristics: *(C1)* the set of properties that contribute in decomposing that property, *(C2)* for

each contributing property, the components that realize it, and *(C3)* a composition function that make it possible to compute the overall value of the property given each contributing property.

To illustrate this, figure 2 gives two examples of compositional properties. Figure 2a describes a compositional relation A.Interface.p=B.Interface.p that links the assembly property `p` of `A` directly to the same property `p` on its subcomponent `B`. The set of the contributing properties of `p` at the level of `A` is the same property `p`. This last property is realized by component `B`, and these two properties are equal (identity function). In figure 2b, the compositional relation A.Interface.p=f(B.Interface.p1,C.Interface.p2) now relates the property `p` of `A` to the two properties `p1` and `p2` ascribed to `B` and `C`. The set of contributing properties of `p` are now `p1` and `p2`; they are respectively realized on components `B` and `C`, and the composition function is `f` (which can be, for example, a simple arithmetical sum or min function).



(a) (b)

Figure 2: Examples of compositional relations.

For a compositional property, determining these *compositional information* formally may be hard, even impossible, and may also require deeper analysis, especially as the composition function (*C3*) may be difficult to express. The fact that our study is restricted both to non-functional properties that are directly derived from single components, and compositional properties, which are a function of properties of its components only (no system environment, or architecture-related factors), simplifies the identification of these information. Even with these hypothesis, there are still many dependencies between properties as well as measurement influences during monitoring (a form of the observer effect, at least because monitoring consumes time and space), which are hard to express and model using simple compositional functions. Compositional functions are to be defined individually for each considered compositional property, and it is necessary to provide a trade-off between simple and provable formula and complicated but more error-prone functions. In our case the composition functions can be defined with almost arbitrary code, as these functions are specified with assertion-based contracts (see section 4.1), so that checking can be performed during both testing and exploitation stages.

We thus suppose that for each compositional property, compositional information are provided through descriptive meta-data that may be expressed using code annotations or *Domain Specific Languages* (DSL) facilities.[2] To enable reasoning on them at runtime, compositional properties are reified as meta-objects. A *CompositionalPropertyManager* (upper left part of figure 3) is built for each component to manage the set of its compositional properties. It uses the provided compositional information to instantiate and register a *CompositionalPropertyObject* (meta-object on upper right part) that both reifies each compositional property and gives access to all of its compositional information: its value, all other properties necessary to compute its composition function and their contributing components. At runtime, the *CompositionalPropertyManager* is exploited by elements at the base or at the meta level to retrieve the corresponding *CompositionalPropertyObject* and get the compositional information about the property it reifies. For example, to get the value of a property (action 1 on figure 3), the implementation of the component function at the base level retrieves the compositional property object from the property manager (2 and 3) and then gets the current value computed from the compositional function (4 and 5). Other compositional information can be retrieved similarly.

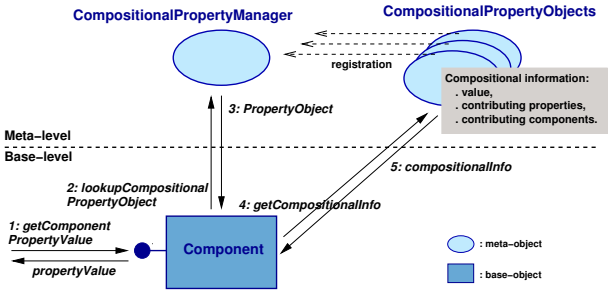


Figure 3: Usage scenario of meta-level elements.

3 Mapping to the Fractal Component Platform

Fractal [5] is a modular and extensible component model with different implementations. It supports composite and shared components (components can be formed from other components and contained in several distinct components), reflection (the execution of components can be observed), and reconfiguration capabilities (component instances can be deployed, removed and replaced dynamically). *Fractal*

[2]For example, our prototype implementation uses Java annotations.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

components are runtime entities that communicate through provided and required interface bindings. Technical aspects can be integrated into components through *controllers*, and some basic controller interfaces are provided to manage interface bindings, lifecycle and content of components (respectively described as `BC`, `LC` and `CC` in figure 4).

3.1 Working Example

We use as a working example an instant messaging server that automatically groups users into chat rooms, and streams videos to grouped users according to their interests. This application is already operational and has been developed using both *Fractal* components and web services. The server, shown on figure 4, is represented by the composite component `FractalInstantCommunication`, which is formed out of three subcomponents: `InstantGroup` manages the users and their grouping through its provided interface `UserMgmt`[3], `VideoService` manages the video streaming service, and `BdwMonitor` monitors the network bandwidth and measures the overall bandwidth consumption of the server. `InstantGroup` is composed of `UserManager` which manages the users, `GroupManager` which manages groups, `MsgMonitor` which monitors messages exchanged between users, and `InstantFacade` which pilots the other components. `InstantGroup` also exhibits the properties `maxUsers` and `groupedUsersRatio` which, respectively, describe the maximum number of concurrent users that the server supports, and the rate of users that have been grouped. `VideoService` is composed of `VideoManager` which manages the video streaming, and `VideoMonitor` which monitors the bandwidth consumption of the video service. The content of `BdwMonitor` is detailed later.

3.2 Integrating Patterns

The *Fractal* component model also proposes control interfaces for attributes in order to model orthogonal properties of components. They give access to component attributes before starting a component and without needing to bind and use its functional interfaces. Moreover, these interfaces offer various access modes (read and/or write accesses), which make possible to respect the difference between properties of nature and functioning properties, which cannot be changed, and configuration parameters which can be modified. Hence, as *properties of nature*, *configuration parameters* and *functioning parameters* of components, are modeled through component attributes, they are basically mapped to *Fractal* attributes and attributes control interfaces (see figures 5a and 5b), with their appropriate read-only or read-and-write operations. As *resources* are modeled by components, they are reified using full-fledged *Fractal* components, which do not provide *a priori* advanced services, except for probing their associated physical resource. Moreover, as resources can be used by several distinct components, *component sharing* is exploited to reflect resource sharing at the component composition level (see figure 5c). It enables one to use a same instance of a reified resource component, in several distinct enclosing components, at different level of hierarchy, while preserving component encapsulation.

[3]For lisibility sake, this interface is not detailed on figure 4.

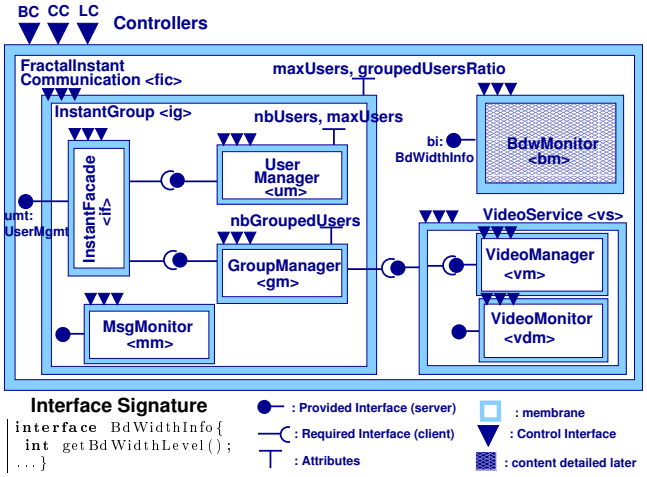


Figure 4: Architecture of the server.

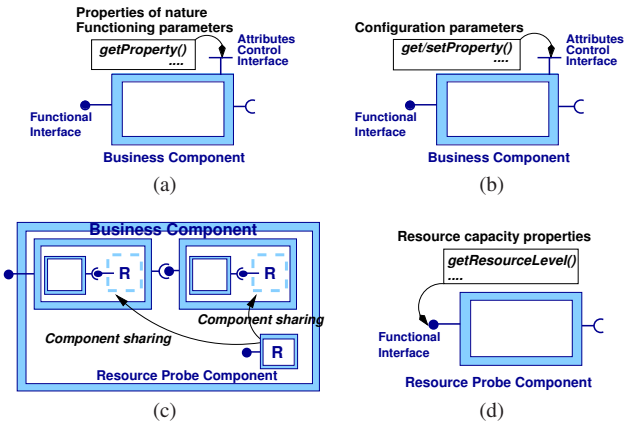


Figure 5: Patterns for Fractal components.

Resource capacity properties represent primitive data collected by resources probes, which are modeled using *Fractal* components. They are thus modeled using functional interfaces attached to the corresponding resource probe component (see figure 5d). The type of collected data

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

is open and to be determined by the developers of probe components. They can span from primitive measurements to more advanced performance indicators processed by various statistical models (interpolations, correlations, etc.). It should be noted that, as *Fractal* components support nesting, the proposed integration patterns can be applied at any level of hierarchy.

In our example (see figure 4), properties are modeled as follows. The `maxUsers` and `groupedUsersRatio` properties of `InstantGroup` are respectively a configuration parameter that is defined when configuring the server to set the maximum threshold of concurrent users, and a functioning parameter that describes the rate of users that have been grouped. They are modeled with read-and-write *Fractal* attributes. The property `nbUsers` of `UserManager` is a functioning parameter that describes how many users have been registered. It is modeled with a read-only *Fractal* attribute. The property `nbGroupedUsers` of `GroupManager` is a functioning parameter that describes how many users have been grouped, and is also modeled with a read-only attribute. All these attributes are accessed through the attributes controller interface of their corresponding component. As for the network bandwidth property, the associated resource probe is modeled with a *Fractal* component (`BdwMonitor`), and the probed data, such as the level of network bandwidth used (`getBdWidthLevel()`), are modeled through the functional interface `BdWidthInfo`. Moreover, as `BdwMonitor` relies on the level of bandwidth used for the messaging and video service to compute the overall bandwidth, it uses `MsgMonitor` and `VideoMonitor` which are then shared and hosted in `BdwMonitor`. For lisibility sake, the sharing of these components is further detailed in section 4.4.

4 Exploitation in Contract Negotiation

In this section, we briefly introduce a contracting system for the *Fractal* platform, and illustrate how the proposed patterns and the reasoning support for compositional properties are used to develop a propagative contract negotiation policy.

4.1 The Contracting System ConFract

ConFract [7] is a contracting system for *Fractal* components, which provides several kinds of contracts, both on interfaces and on components themselves. In *ConFract*, contracts are first class entities during both configuration and run times. They follow the lifecycle of components, and are automatically updated when dynamic reconfigurations occur. Contracts are composed of provisions which are built, at assembly time, from specifications provided by designers. Specifications are currently written in an executable assertion language which is inspired by *OCL* [17] and adapted to the *Fractal* model. They support classic categories of specifications (*pre*, *post*, *inv*, *rely*), but their scope can be both on interfaces and components. The *ConFract* system distinguishes various types of contracts: classic *interface contracts*, similar to object contracts [15], are built on bindings between a required and a provided interface, and different *composition contracts* are built on the external and the internal sides of components, to respectively express the usage and the internal assembly of components.

The various contracts are hierarchically managed at the level of each composite component, by dedicated entities which are implemented in *Fractal* by contract controllers. These controllers also operate contract checkings when appropriate events occur, and, when contracts are violated, they throw an exception describing the context of the violation. By using a metamodel, *ConFract* also assigns, for each category of specifications, appropriate *responsibilities* to involved components, which can notably be *guarantor*, which acts to ensure the provision and must be notified in case of violation of the provision, and *beneficiaries* which can rely on the provision. More details about *ConFract* can be found in [7].

4.2 Negotiation Mechanisms

As contracts can specify non-functional properties that may fluctuate, contracts may be frequently violated, and some mechanisms to handle these violations are required. To this end, a general negotiation model has been proposed [6]. It aims at restoring the validity of violated contracts by activating an *atomic negotiation* for each violated provision of a contract. This negotiation can occur at assembly time if a provision can be statically checked, or dynamically, at run time, if it requires an execution context.

The negotiation model relies on the clearly identified responsibilities (beneficiaries, guarantor) assigned by the *ConFract* system to participating components. It makes negotiating parties interact following a negotiation protocol which is partly inspired from the extended *Contract-Net Protocol* (CNP) [20], commonly applied in multi-agent systems for decentralized tasks allocation. In *ConFract*, the negotiating parties are basically the contract controller and the responsible participating components, which are determined for each violated contract provision. The contract controller acts in the role of the negotiation initiator and conducts the negotiation process, as it manages contract lifecycle and operates contract checking. The protocol basically organizes the interactions between the contract controller and the responsible components following three steps (request of proposals, proposal of modifications and re-checking of the provision against the proposed modification). Finally, the reponsibilities of participating com-

ponents are also exploited to develop different *negotiation policies* which drive the whole negotiation process. Currently, a concession-based policy, in which the negotiation initiator requests concessions from the beneficiary components only, is provided [6].

4.3 A Propagative Negotiation Policy

To enrich the model with more powerful kinds of negotiation, our objective is to design another negotiation policy, which now consists in exploiting the responsibility of the *guarantor* component, and propagating negotiation processes down the hierarchy of the guarantor component. *Some contributing components* at lower levels are then consulted so that they can make some *efforts* to revalidate a violated contract at a higher level. To successfully achieve this, the propagation of the negotiation strongly relies on the compositional information. The contributing properties that decompose a given compositional property are used to drive the propagation from a level of component hierarchy to the sub-level. They are identified using the compositional function. Moreover, for each contibuting property, its realizing component is explicitly identified according to the integration patterns. Each contributing component which is asked for efforts, may then propose some changes regarding the property it realizes.

When the checking of a contract provision fails, the overall negotiation process using the propagative negotiation policy involves the contract controller, which manages the violated contract and the guarantor component. It then executes according to the following steps :

1. The contract controller requests proposals from the guarantor component. In response to these requests, the guarantor component can then either make proposals to revalidate the provision at its level or, decide to consult some components in its content (if it is composite) ;

2. In this latter case, the contract controller of the guarantor takes in charge the negotiation and thus have to identify the set of components that contribute in the property that is specified in the contract provision, and that are to be consulted ;

3. These components either implement the property, or belong to the set of components that contribute in decomposing that property. To precisely identify them, this contract controller then either uses the integration patterns to identify the component that implement the property, or interacts with the compositional properties manager[4] and the associated compositional property meta-object to retrieve the compositional information that describe the decomposition of that property, following the protocol described in figure 3 page 4 ;

4. Once identified, these contributing components are consulted by the contract controller in order to make proposals that may revalidate the violated contract provision. At their turn, they can either propose changes that may revalidate the contract provision, or take in charge the negotiation and propagate it in their content, following the process as in step 2.

[4]The compositional properties manager is implemented in *Fractal* by a dedicated compositional properties controller.

To synthesize this, figure 6 shows an overview of the various entities involved and their interactions during the compositional reasoning and the propagative negotiation process. The negotiation focuses on a contract provision built from the specification C.ci.P < 100, which expresses a maximum threshold for the property P realized through the interface ci of the component C. The contract is managed by the contract controller (CTC) of the component D, which then activates an atomic negotiation in case of violation. The meta-object of the compositional property P is built from the compositional function C.ci.P := f(A.ai.P1,B.bi.P2). It is exploited by the contract controller of the component C to retrieve the compositional information, and identify the set of contributing components, A and B, in order to consult them.

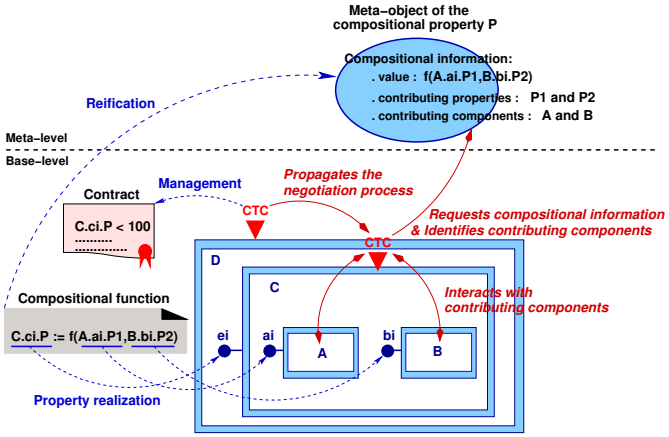


Figure 6: Overview of the propagative negotiation process.

4.4 Illustration

Contracts.

Figure 7 shows an *internal composition contract* which is built in the content of the component `<fic>`. This contract is managed by the contract controller (`CTC` in figure 7) of `<fic>`, and contains three contract provisions which

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

express some internal behavior rules on the implementation of <fic>.

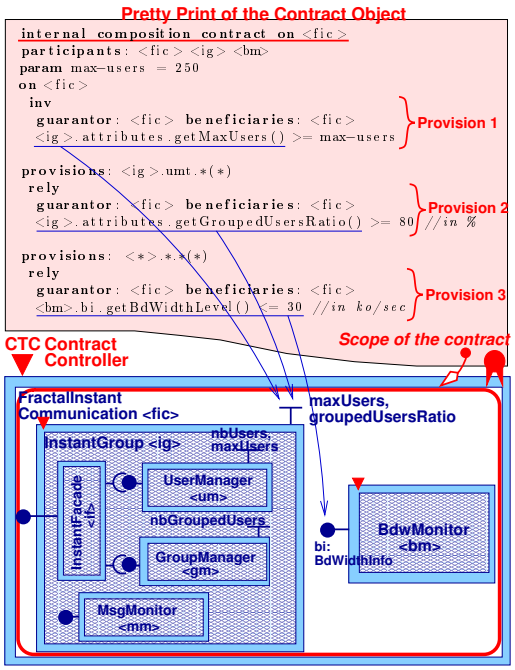


Figure 7: Contracting some properties on the server.

The first provision (see figure 7) defines an invariant on the configuration of <ig>, such that the maximum threshold of concurrent users that the server can support (maxUsers) is higher than 250 users. The second one constraints <ig> by defining a minimum threshold of 80 % for the groupedUsersRatio (on 10 registered users, at least 8 of them must belong to a group), which must hold all along the execution of every method of <umt> (rely construction and operator $*$). The third provision constraints <bm> by defining a bandwidth consumption threshold of 30 ko/sec, which is required to prevent a high bandwidth use. This constraint must hold all along the execution of every method in the content of <fic>. As for their checking, the first contract provision is checked at configuration time, as it specifies an invariant of the configuration. The other provisions, which specify functioning and resource capacity properties, are checked at runtime. Regarding responsibilities, for each of these three contract provisions, <fic> is at the same time, the guarantor and the beneficiary component, as it has to take in charge its internal assembly and also benefits from it[5].

[5]This is a special case of responsibilities attributed for an internal composition contract. For other types of contracts and specification categories, responsibilities are more complex [7].

Scenario for a configuration parameter.

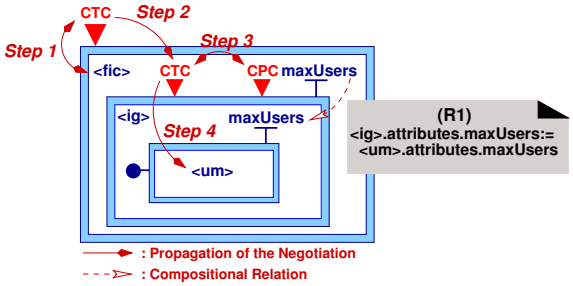


Figure 8: Propagative scheme for the maxUsers property.

The first provision may be violated, if for example, the component <ig> supports by default a maximum threshold of 100 users. Let us suppose that the compositional relation ($R1$) (see figure 8) is provided to describe the fact that the property maxUsers of <ig> decomposes itself identically into the same property maxUsers of <um>. The negotiation process then involves the contract controller (CTC) of <fic> and <fic> itself, as the unique guarantor. It executes as follows. First, the contract controller consults the component <fic> and requests from it some proposals (*step 1*). As <fic> is responsible of its internal assembly, it then takes in charge the negotiation process and consults its subcomponent <ig>, which carries the property maxUsers (*step 2*). To propagate the negotiation in its content, the contract controller of <ig> interacts with the compositional properties controller (named CPC in figure 8) and the compositional property meta-object associated to the maxUsers property, to identify the components to be consulted. The compositional information that describe the maxUsers property (*step 3*) are then retrieved, and the component <um> is identified as the unique contributing component. The contract controller of <ig> requests proposals from <um> which, according to its negotiation reasoning, makes proposals that may, for example, consist in reconfiguring its parameter with a higher value of maxUsers (maxUsers=300 for example) (*step 4*). For each proposal, the contract controller of <ig> uses the compositional function to evaluate whether the proposed changes are sufficient to revalidate the contract provision.

Scenario for a functioning parameter.

The second provision may be violated if the grouped users ratio is lower than 80 %. To negotiate this, let us suppose that the compositional relation ($R2$) (see figure 9), describes the decomposition of the property groupedUsersRatio of <ig> into the property nbGroupedUsers of <gm> and nbUsers of <um>. It expresses the fact that the grouped users ratio is equal to the ratio between the number of users

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

in groups and the overall number of users. The negotiation process involves the same negotiating parties as in the previous example, and it is propagated at the level of `<ig>`, using the same propagation scheme (*step 1* and *2*). However, the components that contribute here in realizing the property `groupedUsersRatio` are `<gm>` and `<um>`, which respectively exhibit the property `nbGroupedUsers` and `nbUsers` (*step 3*). They are thus consulted (*step 4*), but, as these properties describe functioning properties of `<gm>` and `<um>`, they cannot be directly changed. `<gm>` and `<um>` are likely to be unable to propose some efforts. The negotiation then terminates with a failure, which leads to an exception. This exception may be caught outside any negotiation process in order to perform more *adhoc* and global adaptations or reconfigurations of components (changing the `<gm>` component,etc.).

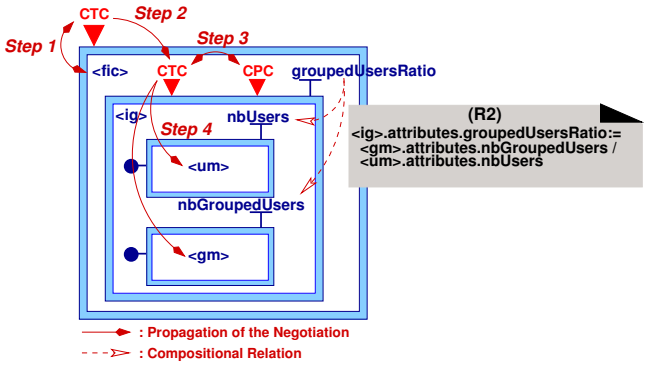


Figure 9: Propagative scheme for the `groupedUsersRatio` property.

Scenario for a resource capacity property.

The third provision is challenged if the global bandwidth consumption exceeds 30 ko/sec. As the `BdwMonitor` component relies on the bandwidth levels of the messaging and video services, which are measured by the probe components `<mm>` and `<vdm>`, these two components are shared and their associated slave instances, `<mm'>` and `<vdm'>`, are hosted in the content of `BdwMonitor`. Besides, the compositional relation named ($R3$) in figure 10 is provided to describe that the property `bdWidthLevel` of `<bm>` decomposes into the sum of the property `bdWidthLevel` of `<mm'>` and `<vdm'>`. The negotiation process is then propagated at the level of `<bm>`. Its contract controller takes in charge the negotiation (*step 2*) and identifies the components `<mm'>` and `<vdm'>` as the contributing components, using the compositional contract controller and the property meta-object (*step 3*). However, as opposed to previous examples, `<mm'>` and `<vdm'>` are not consulted as they merely represent the slave instances of resource probe components, which cannot propose efforts at the application level. The sharing relation is then exploited to retrieve the reference to the master probe components `<mm>` and `<vdm>` (*step 4*), from which the enclosing business components `<ig>` and `<vs>` are retrieved (*step 5*). `<vg>` and `<ig>` are then consulted by the contract controller of `<bm>` (*step 6*), as they are the components at the application-level that may propose efforts to lower the bandwidth consumption (selecting lower bitrates, changing used codecs, compressing file transfers, etc.).

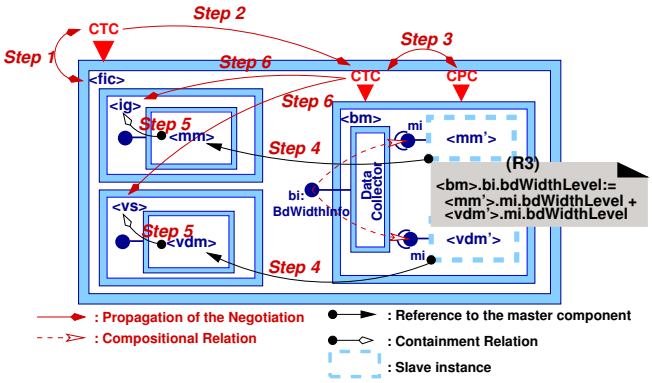


Figure 10: Propagative scheme for the `bdWidthLevel` property.

5 Conclusion

In this paper, we proposed integration patterns that model low-level measurable non-functional properties to individual software components. These patterns are based on a classification that focuses on non-functional properties that derive directly from components, and that considers their relation to components and their lifecycle. Properties are then distinguished among attributes of nature, configurable parameters, functioning parameters, resources and resource capacities. Integration patterns are defined at an abstract level of architecture specification using the *UML*, and we have also described how they map to a hierarchical component platform, namely *Fractal*. As properties are clearly modeled to components and derived directly only from them, some simple compositional relations, which describe the realization of properties given component compositions, can be expressed. Elements at the component meta-level have also been provided to support reasoning on such compositional properties. We finally showed how integration patterns and the compositional support are then further exploited to negotiate non-functional contracts on *Fractal* hierarchical components. They are used in a general propagative scheme, which, by following the compositional relationship between properties, propagate the negotiation to

contributing components, so that they may propose efforts to revalidate violated contracts.

Currently, the classification and integration patterns are limited to the range of low-level measurable properties and have to be developed by considering other non-functional properties such as high-level (maintainability, reusability, availability...) or temporal properties (execution time, latency, periodicity...). To do so, high-level properties should be classified and decomposed into successive lower-level properties that could be measured. However, identifying the relevant non-functional properties is essentially domain-specific, and decomposing them requires a very extensive work, as shown in the QoS properties catalog and in the QoS mapping system developed respectively in the UniFrame [4] and COMQUAD projects [22]. As for temporal properties, they should require further information about the execution behavior, and could be measured using scenario simulation approaches [3] or static timing analysis frameworks [10].

The working example of this paper is extracted from a larger web system, which manages automatic grouping of users and application sharing. This system is already operational and has been developed using the *Fractal* component platform and web service technologies. The integration of the proposed patterns have been validated on this system. The proofs of concept of the compositional meta-level and the propagative scheme regarding contract negotiation have been integrated. Full integration is expected to be finalized soon. Our short term work consists in enhancing and getting more validation elements on the proposed classification and integration patterns, as well as the compositional reasoning support. Other future work will focus on exploiting these patterns in service-oriented architectures built from hierarchical components.

References

[1] M. Barbacci, M. Klein, T. Longstaff, and C. Weinstock. Quality Attributes. Technical Report CMU/SEI-95-TR-021, CMU/SEI, December 1995.

[2] M. Bertoa and A. Vallecillo. Quality Attributes for COTS Components. In *ECOOP'2002 QAOOSE Workshop*. Springer LNCS, 2002.

[3] E. Bondarev, J. Muskens, P. de With, M. Chaudron, and J. Lukkien. Predicting Real-Time Properties of Component Assemblies: A Scenario-Simulation Approach. In *EUROMICRO-SEAA'2004*. IEEE Computer, 2004.

[4] G. Brahnmath, R. Raje, A. Olson, B. Bryant, M. Auguston, and C. Burt. A Quality of Service Catalog for Software Components. In *SESEC'2002*, 2002.

[5] E. Bruneton, T. Coupaye, M. Leclercq, V. Quéma, and J.-B. Stefani. An Open Component Model and Its Support in Java. In *CBSE'2004*. Springer - LNCS, 2004.

[6] H. Chang and P. Collet. Fine-grained Contract Negotiation for Hierarchical Software Components. In *EUROMICRO-SEAA'2005*. IEEE Computer, 2005.

[7] P. Collet, R. Rousseau, T. Coupaye, and N. Rivierre. A Contracting System for Hierarchical Components. In *CBSE'2005*. Springer - LNCS, 2005.

[8] I. Crnkovic, M. Larsson, and O. Preiss. Concerning Predictability in Dependable Component-Based Systems: Classification of Quality Attributes. In *DSN'2004 WADS Workshop*. Springer - LNCS, 2004.

[9] G. Czajkowski, S. Hahn, G. Skinner, P. Soper, and C. Bryce. A Resource Management Interface for the Java Platform. Technical Report SMLI TR-2003-124, Sun Research, 2003.

[10] O. Defour, J.-M. Jézéquel, and N. Plouzeau. Applying CLP to Predict Extra-Functional Properties of Component-Based Models. In *ICLP'2004*. Springer - LNCS, 2004.

[11] E. Eskenazi, A. Fioukov, D. Hammer, and M. Chaudron. Estimation of Static Memory Consumption for Systems Built from Source Code Components. In *EUROMICRO-SEAA'2002*. IEEE Computer, 2002.

[12] G. Blair and A. Andersen and L. Blair and G. Coulson and D. Sanchez. Supporting Dynamic QoS Management Functions in a Reflective Middleware Platform. *IEE Proceedings Software*, 147(1), 2000.

[13] S. Hissam, G. Moreno, J. Stafford, and K. Wallnau. Enabling Predictable Assembly. *Journal of Systems and Software*, 65(3), 2003.

[14] M. Leclercq, V. Quéma, and J.-B. Stefani. DREAM: A Component Framework for Constructing Resource-Aware, Configurable Middleware. *IEEE Distributed Systems Online*, 6(9), 2005.

[15] B. Meyer. Applying "Design by Contract". *IEEE Computer*, 25(10):40–51, Oct. 1992.

[16] N. Wang and D. Schmidt and K. Parameswaran and M. Kircher. Towards a Reflective Middleware Framework for QoS-enabled CORBA Component Model Applications. *IEEE Distributed Systems Online special issue on Reflective Middleware*, 2(5), 2001.

[17] OMG. Object Constraint Language Specification. Technical report, IBM, Sept. 1997.

[18] E. Putrycz and M. Slavescu. Solving Performance Issues in COTS-Based Systems. In *ICCBSS'2006*. IEEE Computer Society, 2006.

[19] R. Reussner, H. Schmidt, and I. Poernomo. Reliability Prediction for Component-based Software Architectures. *Journal of Systems and Software*, 66(3), 2003.

[20] R. G. Smith. The Contract Net Protocol: High-Level Communication and Control in a Distributed Problem Solver. *IEEE Transactions on Computers*, 29:1104–1113, 1980.

[21] K. Wallnau, J. Stafford, S. Hissam, and M. Klein. On the Relationship of Software Architecture to Software Component Technology. In *ECOOP'2001 WCOP Workshop*, 2001.

[22] S. Zschaler. Towards a Semantic Framework for Non-functional Specifications of Component-Based Systems. In *EUROMICRO-SEAA'2004*. IEEE Computer, 2004.

Component-based System Integration via (Meta)Model Composition*

Krishnakumar Balasubramanian, Douglas C. Schmidt, Zoltán Molnár, Ákos Lédeczi
Institute for Software Integrated Systems
Vanderbilt University, Nashville
{kitty,schmidt,zolmol,akos}@isis.vanderbilt.edu

Abstract

This paper provides three contributions to the study of functional integration of distributed enterprise systems. First, we describe the challenges associated with functionally integrating the software of these systems. Second, we describe how the composition of domain-specific modeling languages (DSMLs) can simplify the functional integration of enterprise distributed systems by enabling the combination of diverse middleware technologies. Third, we demonstrate how composing DSMLs can solve functional integration problems by reverse engineering an existing CORBA Component Model (CCM) system and exposing it as Web Service(s) to web clients who use these services. This paper shows that functional integration done using (meta)model composition provides significant benefits with respect to automation, reusability, and scalability compared to conventional integration processes and methods.

1. Introduction

1.1. Challenges of Functional Integration

With the emergence of commercial-off-the-shelf (COTS) component middleware technologies, such as Enterprise Java Beans (EJB), CORBA Component Model (CCM), and Microsoft .NET Framework, software developers are increasingly faced with the task of integrating heterogeneous enterprise distributed systems built using different COTS technologies, rather than just integrating proprietary software developed in-house. Although there are well-documented patterns [8] and techniques [4] for system integration using various component middleware technologies, system integration is still largely a tedious and error-prone manual process. To improve this process, component developers and system integrators must therefore understand key properties of the systems ("system" refers to an enterprise distributed system built using component middleware like EJB, Microsoft .NET, or CCM in the remainder of this paper) they are integrating, as well as the integration technologies they are applying.

This paper describes technologies that help simplify the *functional integration* of systems built using component middleware. This type of integration operates at the logical business layer, typically using distributed objects/components, exposing existing functionality as services, or using messaging middleware. Functional integration of systems is hard due to the variety of available component middleware technologies. These technologies differ in many ways, including the protocol level, the data format level, the implementation language level, and/or the deployment environment level. In general, however, component middleware technologies are a more effective technology base than the brittle proprietary infrastructure used in legacy systems, which have historically been built in a vertical, stove-piped fashion.

Despite the benefits of component middleware, key challenges in functional integration remain unresolved when integrating large-scale systems developed using heterogeneous COTS middleware. These challenges include (1) *integration design*, which involves choosing the right abstraction for integration, (2) *interface mapping*, which reconciles different datatypes, (3) *technology mapping*, which reconciles various low-level issues, (4) *deployment mapping*, which involves planning the deployment of heterogeneous COTS middleware, and (5) *portability incompatibilities* between different implementations of the same middleware technology. The lack of simplification and automation in resolving these challenges significantly hinders effective system integration.

1.2. Solution Approach→Functional Integration using (Meta)Model Composition

A promising approach to address the integration challenges outlined above is *Model-Driven Engineering* (MDE), which involves the systematic use of models as essential artifacts throughout the software lifecycle [20]. At

* This work was sponsored in part by grants from Raytheon and Lockheed Martin Advanced Technology Laboratories.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

the core of MDE is the concept of *domain-specific modeling languages* (DSMLs) [12], whose type systems formalize the application structure, behavior, and requirements within particular domains, such as software defined radios, avionics mission computing, online financial services, warehouse management, or even the domain of middleware platforms.

While DSMLs have been used to help software developers create homogeneous systems [11, 21], enterprise distributed systems are rarely homogeneous. A single DSML developed for a particular component middleware technology, such as EJB or CCM, may therefore not be applicable to model, analyze, and synthesize key concepts of Web Services. To integrate heterogeneous systems successfully, therefore, system integrators need tools that can provide them with a unified view of the entire enterprise system, while also allowing them fine-grained control over specific subsystems and components.

Our approach to integrating heterogeneous systems is *(meta)model composition*[1] [1], which (1) creates a new DSML from multiple existing DSMLs by adding new elements or extending elements of existing DSMLs, (2) specifies new relationships between existing elements, and (3) defines relationships between new and existing elements. This paper describes *System Integration Modeling Language* (SIML), which is our open-source DSML that enables functional integration of component-based systems via *(meta)model composition*. We developed SIML using the Generic Modeling Environment (GME) [13], which is an open-source meta-programmable modeling environment.

2. Functional Integration of Component-based Systems

To motivate the need for model-driven functional integration capabilities, this section describes the challenges in functional integration of component-based systems.

2.1. Functional Integration Challenges

Functional integration of systems is hard and involves activities that map between various levels of abstraction in the integration lifecycle, including design, implementation, and use of tools. We now describe key challenges associated with integrating older component middleware technologies, such as CCM and EJB, with newer technologies, such as Web Services.

Challenge 1. Choosing an appropriate level of integration. As shown in Step 1 of Figure 1, a key activity is to

1 The term "(meta)model" conveys the fact that this composition technique can be applied to both metamodels *and* models.

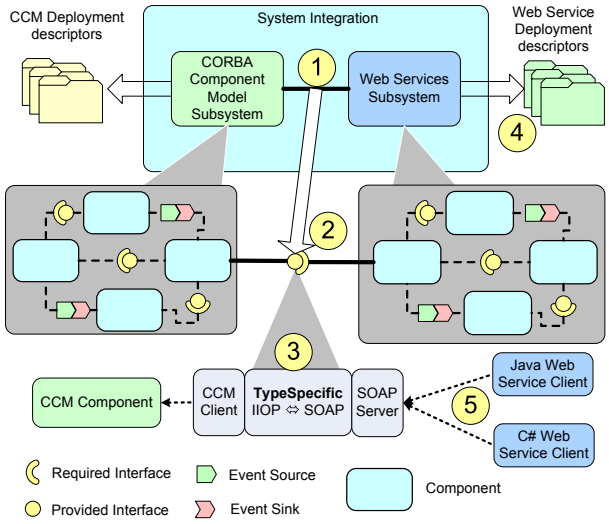


Figure 1. Functional Integration Challenges

identify the right level of abstraction at which functional integration should occur, which involves selecting elements from different technologies being integrated that can serve as conduits for exchanging information. Among the different possible levels at which integration can be performed, criteria for determining the appropriate level of integration include: (1) the number of normalizations, *i.e.*, conversion to/from the native types, required to ensure communication between the peer entities being integrated, (2) the number (and hence the overhead) of as well as the flexibility of deployment, *i.e.*, in-process/out-of-process etc. of runtime entities required to support the functional integration, (3) the number of changes to the integration architecture required corresponding to changes to the peers that are being integrated, (4) available choices of platform-specific infrastructure, *i.e.*, operating systems, programming languages *et al.*, associated with performing integration at a particular level. Attempting integration at the wrong level of abstraction can yield brittle integration architectures that break when changes occur to either the source or target system being integrated.

Challenge 2. Reconciling differences in interface specifications. After the level of abstraction to perform functional integration is determined, it is necessary to map the interfaces exposed by elements of the different technologies as shown in Step 2 of Figure 1. Common COTS middleware technologies usually have an interface definition mechanism that is separate from the implementation details, *e.g.* CCM uses the OMG Interface Definition Language (IDL), whereas Web Services use W3C Web Services Definition Language (WSDL). Irrespective of the mechanism used to define interfaces, mapping of interfaces between any two

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

technologies involves at least three tasks: (1) *datatype mapping*, which involves mapping a datatype (both pre-defined and complex types) from source to target technology, (2) *exception mapping*, which involves mapping exceptions from source to target technology; exceptions are not clubbed together with datatypes since the source or target technologies might not have a notion of exceptions (*e.g*. Microsoft's COM uses a *HRESULT* to convey errors instead of using exceptions), and (3) *language mapping*, which involves mapping datatypes between two technologies while accounting for differences in languages at the same time.[2]

Performing these mappings is non-trivial, requires expertise in both the source and target technologies, and exposes severe scalability problems due to their tedium and error-proneness if they are not automated.

Challenge 3. Managing differences in implementation technologies. The interface mapping described above addresses the high-level details of how information is exchanged between different technologies being integrated. As shown in Step 3 of Figure 1, however, low-level technology details such as networking, authentication and authorization *et al*. are responsible to actually delivering such integration. This involves a technology mapping and includes the following activities: (1) *protocol mapping*, which reconciles the differences between the protocols used for communication between the two technologies, (2) *discovery mapping*, which allows bootstrapping and discovery of components/services between source and target technologies, and (3) *Quality of Service (QoS) mapping*, which maps QoS mechanisms between source and target technologies to ensure that service-level agreements (SLAs) are maintained.

Mapping of protocol, discovery, and QoS technology details requires not only expertise in the source/target technologies, but also intimate knowledge of the implementation details of these technologies.

Challenge 4. Managing deployment of subsystems. Component middleware technologies use declarative notations (such as XML descriptors, source-code attributes, and annotations) to capture various configuration options. Example metadata include EJB deployment descriptors, .NET assembly manifests, and CCM deployment descriptors. As shown in Step 4 of Figure 1, system integrators must track and configure metadata correctly during integration and deployment. In many cases, the correct functionality of the integrated system depends on correct configuration of the metadata.

Challenge 5. Dealing with interoperability issues. Unless a middleware technology has only one version implemented by one provider, there may be multiple implementations from different providers. As shown in Step 5 of Figure 1, differences between these implementations will likely arise due to non-conformant extension to standards, different interpretations of the same (often vague) specification, or implementation bugs. Regardless of the reasons for incompatibility, however, problems arise that often manifest themselves only during system integration. Examples of such differences are highlighted by the presence of efforts like the Web Services-Interoperability Basic Profile (WS-I) [3], which is a standard aimed at ensuring compatibility between the Web Services implementations from different vendors.

Due to the challenges described above, significant integration effort is spent on configuration activities, such as modifying deployment descriptors, and interoperability activities, such as handcrafting protocol adapters to link different systems together, which does not scale up as the number of components in the system increases or the number of adaptations required increases. Problems discovered at integration stage often require changes to the implementation, and thus necessitate interactions between developers and integrators. These interactions are often inconvenient, and even infeasible (especially when using COTS products), and can significantly complicate integration efforts. The remainder of this paper shows how our GME-based (meta)model composition framework and associated tools help address these challenges.

3. DSML Composition using GME

This section describes the (meta)model composition framework in the Generic Modeling Environment (GME) [13]. GME is a meta-programmable modeling environment with a general-purpose editing engine, separate view-controller GUI, and a configurable persistence engine. Since GME is *meta-programmable*, it can be used to design DSMLs, as well as build models that conform to a DSML.

DSMLs are defined by metamodels, hence, DSML composition is defined by (meta)model composition. The specification of how metamodels should be composed, *i.e.*, what concepts in the metamodels that are composed relate to each other and how, can be specified via normal association relationships and additional composition operators, as described in GME [1].

A key property of a composite DSML is that it supports the *open-closed* principle [15], which states that a class should be open for extension but closed with respect to its public interface. In GME, elements of the sub-DSMLs are *closed*,*i.e.*, their semantics cannot be altered in the composite DSML. The composite DSML itself, however, is *open*, *i.e.*, it allows the definition of new interactions and the creation of new derived elements. All tools that are

2 Functional integration is very limited when attempting the latter mapping, which is often done via inter-process communication.

built for each sub-DSML work without any modifications in the composite DSML and all the models built in the sub-DSMLs are also usable in the composite DSML.

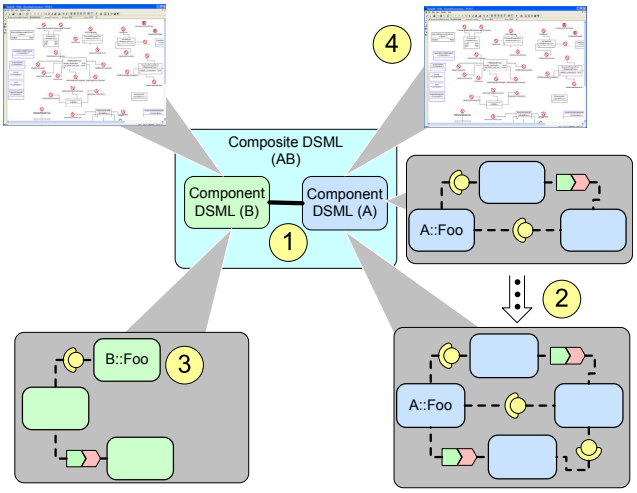


Figure 2. Domain-Specific Modeling Language Composition in GME

We use the following GME (meta)model composition features to support the SIML-based integration of systems built using different middleware technologies, as described in Section 4:

• **Representation of independent concepts.** To enable complete reuse of models and tools of the sub-DSMLs, the composition must be done in such a way that all concepts defined in the sub-DSMLs are preserved. As shown in Step 1 of Figure 2, no elements from either sub-DSMLs should be merged together in the composite DSML. GME's composition operators [1] can be used to create new elements in the composite DSML, but the sub-DSMLs as a whole must remain untouched. As a consequence, any model in a sub-DSML can be imported into the composite language, and vice versa. All models in the composite language that are using concepts from the sub-DSMLs can thus be imported back into the sub-DSML. Existing tools for sub-DSMLs can be reused as well in the composite environment. This technique of composing DSMLs is referred to as *metamodel interfacing* [6] since we create new elements and relationships that provide the interface between the sub-DSMLs.

• **Supporting (meta)model evolution.** DSML composition enables reuse of previously defined (sub-)DSMLs. Just like code reuse in software development, (meta)model reuse can also benefit from the concept of libraries. If an existing (meta)model is simply copied into new composite (meta)models, any changes or upgrades to the original will not propagate to the places where they are used. As shown in Step 2 of Figure 2, if the original (meta)model is imported as a library, GME provides seamless support to update it when new versions become available (libraries are supported in any DSML with GME, not just the metamodeling language) Libraries are read-only projects imported to a host project. Components in the host project can create references to and derivations of library components. The library import process creates a copy of the reused project, so subsequent modifications to the original project are not updated automatically. To update a library inside a host project, a user-initiated refresh operation is required. To achieve unambiguous synchronization, elements inside a project have unique ids, which facilitates correct restoration of all relationships that are established among host project components and the library elements.

• **Partitioning (meta)model namespaces.** When two or more (meta)models are composed, name clashes may occur. To alleviate this problem, (meta)model libraries (and hence the corresponding components DSMLs) can have their own namespaces specified by (meta)modelers, as shown in Step 3 of Figure 2. External software components, such as code generators or model analysis tools that were developed for the composite DSML, must use the fully qualified names. But tools that were developed for component DSMLs will still work because GME sets the context correctly before invoking such a component.

• **Handling constraints.** The syntactic definitions of a metamodel in GME can be augmented by static semantics specifications in the form of Object Constraint Language (OCL) constraint expressions. When metamodels are composed together, the predefined OCL expressions coming from a sub-DSML should not be altered. Therefore GME's Constraint Manager uses namespace specifications to avoid any possible ambiguities, and these expressions are evaluated by the Constraint Manager with the correct types and priorities as defined by the sub-DSML as shown in Step 4 of Figure 2. The composite DSML can also define new OCL expressions to specify the static semantics that augment the specifications originating in the metamodels of the sub-DSMLs.

4. Integrating Systems with SIML

This section describes how we created and applied the *System Integration Modeling Language* (SIML), which is our open-source composite DSML that simplifies functional integration of component-based systems built using heterogeneous middleware technologies.

4.1. The Design and Functionality of SIML

Applying GME's (meta)model composition features to SIML. To support integration of systems built using different middleware technologies, SIML uses the GME (meta)model composition features described in Section 3. SIML is thus a composite DSML that allows integration of systems by composing multiple DSMLs, each representing a different middleware technology. Each sub-DSML is responsible for managing the metadata (creation, as well as generation) of the middleware technology it represents. The composite DSML defines the semantics of the integration, which might include reconciling differences between the diverse technologies, as well as representing characteristics of various implementations. System integrators therefore have a single environment that allows the creation and specification of elements in each sub-DSML, as well as interconnecting them as if they were elements of a single domain.

Applying SIML to compose CCM and Web Services. Our initial use of SIML was to help integrate CCM with Web Services in the context of the shipboard computing case study described in Section 2. The two sub-DSMLs we needed to integrate to support the new requirements described in Section 2 were:

- The **Platform-Independent Component Modeling Language** (PICML), which enables developers of CCM-based systems to define application interfaces, QoS parameters, and system software building rules, as well as generate valid XML descriptor files that enable automated system deployment.
- The **Web Services Modeling Language** (WSML), which enables development of Web Services, and supports key activities in Web Service development, such as creating a model of a Web Services from existing WSDL files, specifying details of a Web Service including defining new bindings, and auto-generating artifacts required for Web Service deployment.

Since SIML is a composite DSML, all valid elements and interactions from both PICML and WSML are valid in SIML. It is therefore possible to design both CCM components (and assemblies of components), as well as Web Services (and federations of Web Services) using SIML, just as if either PICML or WSML were used independently. The whole is greater than the sum of its parts, however, because SIML defines new interactions that allow connecting a CCM component (or assembly) with a Web Service and automates generation of necessary gateways, which are capabilities that exist in neither PICML nor WSML.

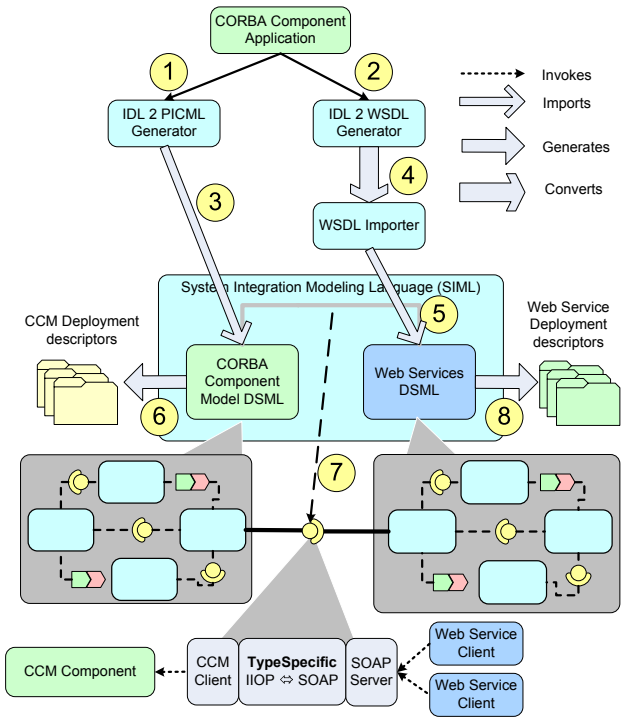


Figure 3. Generating a Web Service Gateway Using SIML

4.2. Resolving Functional Integration Challenges using SIML

We now show how we applied SIML to resolve the functional integration challenges discussed in Section 2.1. Although we focus on the initial version of SIML that supports integration of CCM and Web Services, its design is sufficiently general that it can be applied to integrate many other middleware technologies without undue effort. Figure 3 shows how SIML resolves the following challenges to generate a gateway given an existing CCM application:

Resolving challenge 1. Choosing an appropriate level of integration. As mentioned in Section 2.1, determining the right level of integration requires expertise in all the different technologies being integrated. To allow interactions between CCM components and Web Services, SIML defines interactions between ports of CCM components and ports exposed by the Web Services. Since SIML also automates the generation of the glue code, some choices with respect to the level of integration, *e.g.*, mapping of a CCM port to a Web Service port, are pre-determined, while other decisions, *e.g.*, aggregation of more than one CCM component into a single Web Service, are customizable. SIML thus extends the list of valid interactions of both CCM com-

ponents and Web Services, which is an example of a composite DSML defining interactions that does not exist in its sub-DSMLs. SIML can also partition a large system into hierarchies via the concept of "modules," which can be either CCM components (and assemblies of CCM components) or Web Services. SIML's architecture can be enhanced to support integration of many middleware technologies, by extending the list of interactions defined by SIML to integrate new technologies. For example, SIML could be extended to support interactions between CCM and EJB, or even between Web Services and EJB. Extending SIML to support other technologies, *e.g.* EJB, requires specification of a DSML that describes the elements and interactions of EJB. Once the DSML for EJB is specified, it can be imported into SIML as a library while also assigning a new namespace to it; the creation of a new namespace prevents any clash between the type systems *e.g.*, between a CCM component and EJB component. Interactions between elements of CCM and EJB can then be defined in the composite DSML. From these new interactions, generative techniques (as explained in resolution to Challenge 3 below) can be applied to automate the task of integration.

Resolving challenge 2. Reconciling differences in interface specifications. To map interfaces between CCM and Web Services, SIML provides a tool called IDL2WSDL, which automatically converts any valid CORBA IDL file to a corresponding WSDL file. As part of this conversion process, IDL2WSDL performs both *datatype mapping*, which maps CORBA datatypes to WSDL datatypes, and *exception mapping*, which maps both CORBA exceptions to WSDL faults. System integrators are therefore relieved from the intricacies of the mapping. As shown in Figure 3, both IDL and WSDL can also be imported into the DSML environment corresponding to CCM (PICML) and Web Services (WSML), allowing integrators to define interactions between CCM components and Web Services. SIML also supports *language mapping* between ISO/ANSI C++ and Microsoft C++/CLI, which is the .NET framework extension to C++. SIML therefore automates much of the tedious and error-prone details of the interface mapping, thereby allowing system integrators to focus largely on the business logic of the application being integrated.

Resolving challenge 3. Managing differences in implementation technologies. While the rules defined in SIML allow definition of interaction at the modeling level, this feature is not very useful if these definitions cannot be translated into runtime entities that actually perform the interactions. SIML therefore generates *resource adapters*, which automatically convert SOAP requests into IIOP requests, and vice-versa. A *resource adapter* in SIML is implemented as a gateway. SIML allows system integrators to define connections between ports of a CCM component and a Web Service, as shown in Figure 3. These connections are then used by a *model interpreter*, which automatically determines the operation/method signatures of operations/methods of the ports on either end of a connection, and uses this information to automatically generate a gateway. The generated gateway contains all the "glue code" necessary to perform datatype mapping, exception mapping, and language mapping between CCM and Web Services. The gateway generator is configurable and can currently generate Web Service gateways for two different implementation of Web Services: GSOAP [24] and Microsoft ASP.NET. The generated gateway also performs the necessary *protocol mapping* (*i.e.*, between IIOP and SOAP) and *discovery mapping* (*i.e.* automatically connecting to a Naming Service to obtain object references to CCM components). Our initial implementation does not yet support *QoS mapping*, which is the focus of future work, as described in Section 6.

Resolving challenge 4. Managing deployment of subsystems. After the necessary integration gateways have been generated, system integrators also need to deploy and configure the application and the middleware using a variety of metadata in the form of XML descriptors. Since SIML is built using (meta)model composition it can automatically use the tools developed for the sub-DSMLs (*i.e.*, PICML to handle deployment of CCM applications and WSML to handle deployment of Web Services) directly from within SIML.

By encapsulating the required resource adapters inside a a Web Service or CCM component, SIML allows reuse of deployment techniques available for any given middleware system. System integrators therefore do not need to deploy resource adapters separately. While this approach works for in-process resource adapters (such as those generated by SIML), out-of-process resource adapters need support from a deployment descriptor generator. Since SIML is a DSML itself, this support could be added to SIML so it can generate deployment support for out-of-process resource adapters.

Resolving challenge 5. Dealing with interoperability issues. Since knowledge of the underlying middleware technologies is built into SIML, it can automatically compensate for incompatibilities during design time. For example, IDL2WSDL allows generation of WSDL that supports SOAP RPC encoding or an interoperable subset defined in the WS-I Basic Profile. System integrators therefore are better prepared to handle incompatibilities that only show up during integration testing. SIML can also define constraints on WSDL definition as prescribed by the WS-I Basic Profile, so that violations can also be checked at modeling time. Similarly, gateway generation can add workarounds for quirks of particular implementations automatically, thereby relieving system integrators from finding these problems during final integration testing. The automation of gateway generation also scales the integration activity since develop-

Integration Activity	Supported?	Automated?	Level of Automation (# of distinct steps)					
			Using SIML			Using Native Tools		
			Design	Implementation	Tool Use	Design	Implementation	Tool Use
Integration Design	Yes	No	0	0	1	1	1	0
Interface Mapping								
DataType Mapping	Yes	Yes	0	0	1	1	1	0
Exception Mapping	Yes	Yes	0	0	1	1	1	0
Language Mapping	Yes	Yes	0	0	1	1	1	0
Technology Mapping								
Protocol Mapping	Yes	Yes	0	0	1	1	1	0
Discovery Mapping	Yes	Yes	0	0	1	1	1	0
QoS Mapping	No	No	1	1	0	1	1	0
Deployment Mapping								
Descriptor Generation	Yes	Yes	0	0	1	0	0	1
Gateway Placement	No	No	1	1	0	1	1	0
Interoperability Mapping	Yes	Yes	0	0	1	0	1	0

Table 1. Evaluating Functional Integration using SIML

ers need not write system specific integration code. In addition, SIML allows evolution of the integrated system by incrementally adding more components, or targeting different middleware implementations as future needs dictate.

4.3. Evaluating SIML

To evaluate the benefits of SIML, we first define a taxonomy for evaluating technologies that assist the functional integration of CCM and Web Services. We then use this taxonomy to compare SIML with tools that are supplied by vendors for either technology, referred to in Table 1 as *Native tools*. Examples of native tools include the Microsoft Visual Studio and the IBM Eclipse suite, which developers using middleware technologies like .NET and EJB are likely to use. This table depicts the different mapping activities described in Section 4.2 that are typical in functional integration of middleware systems. For each activity the table describes the level of support in SIML and whether the activity is automated. It also describes the level of automation measured as the number of distinct steps performed by a system integrator. Table 1 further decomposes the level of automation into three broad categories: (1) *design*, which denotes that system integrators need to perform a design activity that might include domain analysis, requirement analysis, etc., (2) *implementation*, which denotes that system integrators need to implement some functionality usually by writing code, and (3) *tool use*, which denotes that a tool needs to be used by the system integrators to perform that activity. This categorization assigns a weight commensurate to the skills of the individual responsible for carrying out the task in a particular organization.

Our taxonomy also assumes that design and implementation are orders of magnitude more difficult/time-consuming than tool use. In Table 1, therefore, multiple activities of the same category are considered equal, since the magnitude difference will likely dwarf any small number of steps of any particular category. Thus the table uses 1 to indicate one or more, *i.e.*, $1 \ldots n$ steps, and 0 to indicate that the effort is automated. To estimate the amount of effort required, we sum up each of the three columns (*i.e.*, design, implementation, and tool) and then multiply the result by the weight assigned to each category. For example, a reasonable assignment of weight for these activities might be 10, 5 and 1, for each of design, implementation and tool use. With this assignment, we can see that using SIML requires $2 \times 10 + 2 \times 5 + 8 \times 1 = 38$ distinct steps to achieve functional integration. In comparison, using just the native tools would result in $8 \times 10 + 9 \times 5 + 1 \times 1 = 126$ distinct steps to achieve the same. It should be noted that the number of steps will get reduced drastically as (and when) native tools add support for integration activities.

The numbers in Table 1 are for each unique unit of work per unique pair of source and target technologies, *i.e.*, for a single datatype mapping, a single exception mapping, a single protocol mapping. To calculate the total cost of integration, we must take into account both the number of distinct types/exceptions/languages, and the number of unique pairs of technologies being integrated.

Since SIML allows hierarchical composition of the integration infrastructure, the integration architecture scales along with the increase in the number of components. While the generative techniques applied to generate the gateways scale with the number of components in the system, when the number of components increases to thousands of components, the limitations of visual design tools tend to show up. To overcome the issues with scalability of modeling techniques, we have applied techniques like aspect-oriented weaving of domain-specific models [2] in prior efforts. Such techniques can be applied to automate the modeling activities in SIML in the presence of large number of components, since SIML itself is a domain-specific language for integration.

Table 1 shows that SIML helps reduce the effort by re-

ducing the design and/or implementation activities associated with integration to ordinary tool usage activities. For example, SIML effectively reduces the design and implementation effort required to perform the datatype, exception and language mapping, to a single step of tool use. This table also shows that similar gains can be achieved for complex tasks, such as protocol mapping (conversion between IIOP and SOAP in this case) and discovery mapping (conversion between CORBA Object References and Web Service URIs). Finally, the table reveals current gaps in our toolchain, *i.e.*, SIML does not perform QoS mapping or help with placement of resource adapters (or gateways), which remains as future work.

5. Related Work

This section surveys the technologies that provide the context of our work on system integration in the domain of large-scale distributed enterprise systems. We classify techniques and tools in the integration space according to the role played by the technique/tool in system integration.

Integration evaluation tools enable system integrators to specify the systems/technologies being integrated and evaluate the integration strategy and tools used to achieve integration. For example, IBM's WebSphere [9] supports modeling of integration activities and runs simulations of the data that is exchanged between the different participants to help predict the effects of the integration. While these tools help identify potential integration problems and evaluate the overall integration strategy, they do not replace the actual task of integration itself since these tools use simulation-/emulation-based abstractions of the actual systems. SIML's role is complementary to these tools: once the integration evaluation has been done using these tools, SIML can be used to design the integration, as well as generating the various artifacts required for integration.

Integration design tools. OMG's UML profile for Enterprise Application Integration (EAI) [18] defines a Meta Object Facility (MOF) [17] based metamodel for collaboration modeling, as well as activity modeling. MOF provides facilities for modeling the integration architecture focusing on connectivity, composition and behavior. The EAI UML profile also defines a MOF-based standardized data format to be used by the different systems to exchange data during integration, which is achieved by defining an EAI application metamodel that handles interfaces and metamodels for programming languages such as C, C++, PL/I and COBOL, to aid the automation of transformation. While standardizing on MOF is a step in the right direction, the lack of widespread support for MOF by various tools, and the differences between versions of XML Metadata Interchange (XMI) support in tools lead to problems in practice. Existing integration design tools provide limited support for interface mapping by generating stubs and skeletons for facilitating interface mapping, and perform protocol mapping. However, key activities like discovery mapping, and deployment mapping still needs to programmed by the system integrator. Thus the primary difference between SIML and these tools is that SIML not only allows such integration design, but also automates the generation of key integration artifacts, such as gateways, reducing the amount of effort required to develop and deploy the systems.

Integration patterns [23] provides guidance to system integrators in the form of best patterns and practices with examples of using a particular vendor's products. [8] catalogs common integration patterns with an emphasis on system integration via Message-Oriented Middleware (MOM) using different commercial products. These efforts do not directly provide tools for integration, but instead provide critical guidance to using existing tools to achieve integration. We are enhancing SIML to support modeling integration patterns and using them to enhance the generative capabilities of SIML to enable widely-accepted solutions to common integration problems.

Resource adapters are used during integration to transform data and services exposed by service producers to a form that is amenable to service consumers. While existing standards (such as the Java Messaging Specification [22] and J2EE Connector Architecture Specification [16]) and tools (such as IBM's MQSeries [10]) provide the architectural framework for performing the required adaptations, these tools approach the integration from a middleware and programming perspective, *i.e.*, system integrators are still required to handcraft the "glue" code that invokes the resource adapter frameworks to connect system components together. In contrast, SIML uses syntactic information present in the DSMLs to automatically perform the required mapping/adaptation by generating the necessary "glue" code, and relies on user input only for tool use.

Integration frameworks. Composition in the context of the semantic web and the Web Ontology Language (OWL) [5] has focused on composition of services from unambiguous, formal descriptions of capabilities as exposed by services on the web. Research on service composition has focused on automation and dynamism [19], optimizing the composition such that it is QoS-aware [26], as well as integration on large-scale "system-of-systems" like the GRID [7]. Since these automated composition techniques rely on unambiguous, formal representations of capabilities, system integrators need to make their legacy systems available as Web Services or provide alternate formal mappings of capabilities of the system to be integrated, which may not always be feasible. Our approach to (meta)model composition, however, is not restricted to a single domain, though the semantics are bound at design time. While both ap-

proaches rely on metadata, SIML's use of metadata focuses on the generative capabilities possible rather than on the semantic knowledge extracted from metadata.

Integration quality analysis. As the integration process evolves, it is necessary to validate whether the results are satisfactory from functional and QoS perspectives. Research on QoS issues associated with integration has yielded languages and infrastructure for evaluating Service-Level Agreements (SLAs). Examples include the Web Service Level Agreement language (WSLA) [14] framework, which defines an architecture to define service-level agreements using an XML Schema, and provides associated infrastructure to monitor the conformance of the running system to the desired SLA. Other efforts have focused on defining processes for distributed continuous quality assurance [25] of integrated systems to identify the impact on performance during system evolution. Information from these analysis tools should be incorporated into future integration activities. While these tools can be used to provide input to design-time integration activities, they themselves do not support automated feedback loops. We are adding support for modeling SLAs in SIML to allow evaluation of SLAs before/after integration.

6. Concluding Remarks

The development of enterprise distributed systems increasingly involves more integration of existing COTS software and less in-house development from scratch. With the increase in capabilities of COTS component middleware technologies, the complexity of integration of systems built upon such frameworks is also increasing. This paper shows how a model-driven approach to functional integration of component middleware technologies enhances conventional approaches to system integration, which are tedious, error-prone, and non-scalable for enterprise distributed systems. We then show how DSMLs and (meta)model composition can help to address these limitations.

To demonstrate the viability of our approach, we enhanced support for composition of DSMLs in GME. Using this new capability, we developed the *System Integration Modeling Language* (SIML), which is a DSML composed from two other DSMLs, the CCM profile of Platform-Independent Component Modeling Language (PICML) and the Web Services Modeling Language (WSML). Finally, we evaluated the benefits of our approach by generating a gateway from the model, which automates key steps needed to functionally integrate CCM components with Web Services.

The following is a summary of lessons learned thus far from our work applying (meta)model composition to integrate heterogeneous middleware technologies:

• **Integration tools are becoming as essential as design tools.** SIML is designed to bridge the gap between existing *component technologies* (in which the majority of software systems are built) and *integration middleware* (which facilitate the integration of such systems). SIML elevates the activity of integration to the same level as system design by providing tools which allow integration design of systems built using heterogeneous middleware technologies. Since SIML is a DSML, it can potentially be used as the infrastructure to define constraints on the actual integration process itself, thereby allowing evaluation of service-level agreements prior to the actual integration itself.

• **Automating key portions of the integration process is critical to building large-scale distributed systems.** Compared with conventional approaches, our model-driven approach to system integration automates key aspects of system integration, including gateway "glue code" generation, metadata management, and design-time support for expressing unique domain and/or implementation assumptions. It supports seamless migration of existing investment in models and allows incremental integration of new systems. Moreover, our model-driven approach is general-purpose and can be applied to tool-chains other than GME, as well as help integrate systems other than CCM or Web Services.

• **QoS integration is a complex problem, and requires additional R&D advances.** Though SIML helped map functional aspects of a system from a source technology to a target technology, our work is not complete until the non-functional QoS-related aspects of a system also map seamlessly. For example, technologies like the Real-time CORBA Component Model (RT-CCM) [25] support many QoS-related features (such as thread pools, lanes, priority banded connections, and standard static/dynamic scheduling services) that allow system developers to configure the middleware to build systems with desired QoS features. When systems based on RT-CCM are integrated with other technologies, it is critical to automatically map the QoS-related features used by an application in the source technology to the set of QoS features available in the target technology. For example, a number of specifications have been released for Web Services that target QoS features, such as reliable messaging, security, and notification. The focus of our future efforts in integration involves extending SIML to automatically map QoS features from one technology to another using DSMLs, such that the integration is automated in all aspects – both functional and non-functional.

Instructions for downloading SIML and GME are available at `www.dre.vanderbilt.edu/cosmic`.

References

[1] Ákos Lédeczi, G. Nordstrom, G. Karsai, P. Volgyesi, and M. Maroti. On Metamodel Composition. In *Proceedings of the 2001 IEEE International Conference on Control Applications (CCA)*, pages 756–760, Mexico City, Mexico, 2001. IEEE.

[2] K. Balasubramanian, A. Gokhale, Y. Lin, J. Zhang, and J. Gray. Weaving Deployment Aspects into Domain-Specific Models. *International Journal on Software Engineering and Knowledge Engineering (IJSEKE)*, 16(3), June 2006.

[3] K. Ballinger, D. Ehnebuske, C. Ferris, M. Gudgin, C. K. Liu, M. Nottingham, and P. Yendluri. WS-I Basic Profile. `www.ws-i.org/Profiles/BasicProfile-1.1.html`, April 2006.

[4] C. Britton and P. Bye. *IT Architectures and Middleware: Strategies for Building Large, Integrated Systems*. Addison-Wesley Professional, May 2004.

[5] W. W. W. Consortium. Web Ontology Language. `www.w3.org/2004/OWL/`, Feb 2004.

[6] M. Emerson and J. Sztipanovits. Techniques for metamodel composition. In *The 6th OOPSLA Workshop on Domain-Specific Modeling, OOPSLA 2006*, Portland, OR, Oct 2006. ACM.

[7] I. Foster, C. Kesselman, J. M. Nick, and S. Tuecke. Grid Services for Distributed System Integration. *Computer*, 35(6):37–46, 2002.

[8] G. Hohpe and B. Woolf. *Enterprise Integration Patterns: Designing, Building, and Deploying Messaging Solutions*. Addison-Wesley Professional, October 2003.

[9] IBM. WebSphere. `www.ibm.com/software/info1/websphere/index.jsp`.

[10] IBM. MQSeries Family. `www-4.ibm.com/software/ts/mqseries/`, 1999.

[11] G. Karsai, S. Neema, B. Abbott, and D. Sharp. A Modeling Language and Its Supporting Tools for Avionics Systems. In *Proceedings of 21st Digital Avionics Systems Conf.*, Aug. 2002.

[12] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-Integrated Development of Embedded Software. *Proceedings of the IEEE*, 91(1):145–164, Jan. 2003.

[13] A. Ledeczi, A. Bakay, M. Maroti, P. Volgysei, G. Nordstrom, J. Sprinkle, and G. Karsai. Composing Domain-Specific Design Environments. *IEEE Computer*, pages 44–51, November 2001.

[14] H. Ludwig, A. Keller, A. Dan, R. P. King, and R. Franck. Web Service Level Agreement Language Specification. `researchweb.watson.ibm.com/wsla/documents.html`, January 2003.

[15] B. Meyer. Applying Design By Contract. *Computer (IEEE)*, 25(10):40–51, Oct. 1992.

[16] S. Microsystems. J2EE Connector Architecture Specification. `java.sun.com/j2ee/connector/`, November 2003.

[17] Object Management Group. *MetaObject Facility (MOF) 2.0 Core Specification*, OMG Document ptc/03-10-04 edition, Oct. 2003.

[18] Object Management Group. *UML Profile for Enterprise Application Integration (EAI)*, omg document formal/04-03-26 edition, March 2004.

[19] S. R. Ponnekanti and A. Fox. SWORD: A Developer Toolkit for Web Service Composition. Jan. 01 2002.

[20] D. C. Schmidt. Model-Driven Engineering. *IEEE Computer*, 39(2):41–47, 2006.

[21] J. A. Stankovic, H. Wang, M. Humphrey, R. Zhu, R. Poornalingam, and C. Lu. VEST: Virginia Embedded Systems Toolkit. In *Proceedings of the IEEE Real-time Embedded Systems Workshop*, London, UK, Dec. 2001. IEEE.

[22] SUN. Java Messaging Service Specification. `java.sun.com/products/jms/`, 2002.

[23] D. TrowBridge, U. Roxburgh, G. Hohpe, D. Manolescu, and E. G. Nadhan. Integration Patterns. `msdn.microsoft.com/library/default.asp?url=/library/en-us/dnpag/html/intpatt.asp`, June 2004.

[24] R. van Engelen and K. Gallivan. The gSOAP Toolkit for Web Services and Peer-to-Peer Computing Networks. In *CCGRID*, pages 128–135. IEEE Computer Society, 2002.

[25] N. Wang and C. Gill. Improving Real-time System Configuration via a QoS-aware CORBA Component Model. In *Hawaii International Conference on System Sciences, Software Technology Track, Distributed Object and Component-based Software Systems Minitrack, HICSS 2004*, Kona, HW, Jan. 2004. HICSS.

[26] L. Zeng, B. Benatallah, A. H. Ngu, M. Dumas, J. Kalagnanam, and H. Chang. QoS-Aware Middleware for Web Services Composition. *IEEE Trans. Softw. Eng.*, 30(5):311–327, 2004.

COTS Selection: Past, Present, and Future

Abdallah Mohamed[1], Guenther Ruhe[1], Armin Eberlein[2]
[1]*University of Calgary, 2500 University Drive, NW, Calgary, AB, T2N1N4, Canada*
[2]*American University of Sharjah, Sharjah, P.O. Box 26666, UAE*
{asamoham, ruhe, eberlein}@ucalgary.ca

Abstract

Commercial Off-The-Shelf (COTS) products are increasingly being used in software development. In COTS-based development, selecting appropriate COTS is the most crucial phase. This paper explores the evolution of COTS selection practices, and surveys eighteen of the most significant COTS selection approaches. The paper traces how each approach contributed to the improvement of current COTS selection practices, and then compares them .The paper also highlights some open research issues relevant to the selection process, and concludes with a discussion of possible future directions to address these issues.

1. Introduction

Most of today's software systems include one or more COTS (commercial off-the-shelf) products. COTS are pieces of software that can be reused by software projects to build new systems [1, 2]. Such COTS include word processors, email packages, etc [3].

Performing a good COTS selection plays a critical role in the success of the final system [4]. COTS selection is the process of determining the fitness-of-use of COTS products in a new context, and then selecting one or more products with the highest fitness [5].

COTS selection involves many challenges such as the high complexity of the process [6]. To overcome these challenges, several efforts were made during the last decade to model the COTS selection process. However, none of these methods can be considered as the silver-bullet to solving the COTS selection problem. Different approaches are of different effectiveness and might be suitable for different contexts.

In this paper, we show how different COTS selection approaches contributed to the advance of COTS selection practices. The paper reviews current COTS selection approaches, discusses their contribution, compares them, shows their pros and cons, and briefly explains their main activities. Next, the paper highlights those issues that were not sufficiently addressed by existing approaches, and shows possible future directions that can be used to solve these issues. To the best of our knowledge, there is no previous such thorough surveys.

This paper is structured as follows: Section 2 describes the COTS selection process. The section starts by describing the so-called "*general COTS selection process*" which highlights the main activities that most existing approaches use. Then, we discuss the evaluation process which is the core of any COTS selection approach. Section 3 presents COTS selection approaches and summarizes their contribution to the evolution of COTS selection practices. Section 4 presents some of the 'present' open issues relevant to the COTS selection process. Then, in section 5 some 'future' directions hat can be taken to address these issues are discussed. Finally, conclusions are given in section 6. The paper includes one appendix which describes the approaches listed in section 3.

2. The COTS selection process

2.1. The general COTS selection process

Although there is no commonly accepted method for COTS selection [6], all methods share some key steps that can be iterative and overlapping. These steps formulate what we refer to as the *General COTS Selection (GCS)* process which is described as follows:

Step$_1$: Define the evaluation criteria based on stakeholders' requirements and constraints.

Step$_2$: Search for COTS products.

Step$_3$: Filter the search results based on a set of 'must-have' requirements. This results in defining a short list of most promising COTS candidates which are to be evaluated in more detail.

Step$_4$: Evaluate COTS candidates on the short list

Step$_5$: Analyze the evaluation data (i.e. the output of *Step*$_4$) and select the COTS product that has the best fitness with the criteria. Usually, decision making techniques, e.g. analytic hierarchy process (AHP) [7], are used for making the selection.

After *Step*$_5$, the selected COTS product is usually customized (aka tailored) as needed in order to reduce the mismatches it has with the requirements. A COTS product can be customized in different ways, such as using add-ons, adjusting parameters, etc.

2.2. COTS evaluation

COTS evaluation is the core of the COTS selection process as it is the activity in which the fitness-of-use

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

of COTS products is determined. COTS evaluation provides information necessary for the decision maker to select the best COTS product from a set of competing alternatives [8, 9].

COTS are evaluated against a set of criteria that represents the stakeholders' requirements and system constraints. Kontio et al. [10] suggest to hierarchically define the evaluation criteria, where a set of high-level goals are gradually refined based on such factors as the application requirements and architecture, the COTS capabilities, etc. Maiden et al. [4] agrees with Kontio and further suggest defining the evaluation criteria while at the same time evaluating existing COTS. The technical literature also includes efforts to explain how to define the evaluation criteria based on quality models. For example, Franch et al. [11] and Carvallo et al [12] propose a six-step method to build a structured quality model for the purpose of COTS evaluation. They rely on the ISO/IEC 9126 quality model [13] and further explain activities to define a set of metrics that can be used during the evaluation.

Generally, there are three strategies which can be applied to evaluate COTS products [14, 15]:

1. *Progressive filtering,* which starts with a large number of COTS, and then progressively defines discriminating criteria through successive iterations of product evaluation cycles, where in each cycle 'less fit' products are eliminated. This strategy requires running steps 1 to 4 in the GCS process iteratively until a small number of most promising COTS products is identified from which one or more can be selected for integration into the system.
2. *Puzzle assembly*, which assumes that a COTS-based system requires fitting various components together like pieces of a puzzle. This implies that a product that 'fits' in isolation might not be acceptable when combined with other products. Therefore, this strategy suggests considering the requirements of each product while simultaneously remembering the requirements of other products in the puzzle.
3. *Keystone identification*, which starts by identifying a key requirement (e.g. vendor location or type of technology), and then searching for products that satisfy this keystone requirement. This allows quick elimination of a large number of products that do not satisfy the key requirement.

More than one of the above strategies may be employed in the same project [15]. For example, a developer might use 'keystone identification' first and then later 'progressive filtering'.

3. COTS selection approaches

This section surveys and compares current COTS selection approaches. The section tries to trace how each approach contributed to the improvement of current COTS selection practices. Each approach is discussed in more detail in Appendix 1.

3.1. The evolution of COTS selection practices

Figure 1 traces the improvements made to the COTS selection process over the last decade. One of the first proposals was given by Kontio et. al. [16] who proposed the OTSO (Off-The-Shelf Option) approach for COTS selection in 1995. OTSO is considered an important milestone in the evolution of COTS selection practices as it served as a basis for other approaches. OTSO defined the basic structure of the COTS selection process. This structure is very similar to the GCS process described in section 2.1.

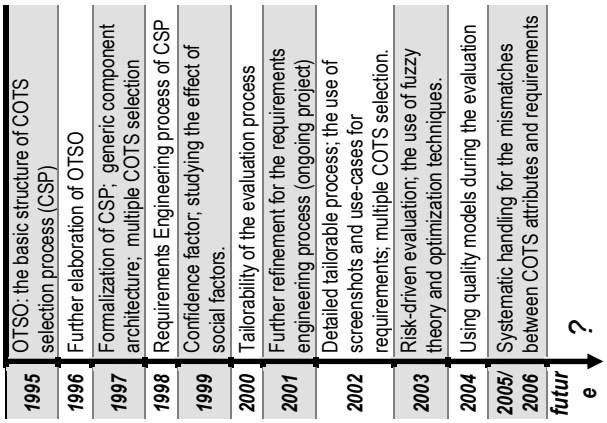


Figure 1 Evolution of COTS selection practices

In 1996, Kontio published several follow-up papers to elaborate OTSO (e.g. [10]) in which the process of defining the evaluation criteria is described in detail.

In 1997, several approaches were proposed:

(i) The IusWare (IUStitia softWARis) [17] approach tried to formalize the selection process, and to address quality requirements during the evaluation process.

(ii) The PRISM (Portable, Reusable, Integrated, Software Modules) [18] approach proposed a generic component architecture that can be used during the COTS evaluation process.

(iii) The CISD (COTS-based Integrated Systems Development) [19] approach proposed a model that can be used to select multiple homogeneous COTS products.

However, it was not until 1998 that another important milestone was reached with the PORE approach [4]. The importance of PORE is that it proposed a requirements engineering process for COTS-based development. PORE suggested that requirements should be elicited and analyzed at the same time when the COTS products are evaluated.

In 1999, several approaches were proposed:

(i) The CEP (Comparative Evaluation Process) approach introduced the use of the so-called confidence factor (CF). The more reliable the source of data, the higher a CF value that source gets. Any estimate we make should be adjusted based on the CF value of the source based on which these estimations are made.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

(ii) The STACE (Social-Technical Approach to COTS Evaluation) approach [20] emphasized the importance of non-technical issues, e.g. social, human, and organizational characteristics, during the evaluation process.
(iii) The CRE (COTS-based RE) approach emphasized the importance of non-functional requirements (NFR) as decisive criteria when comparing COTS alternatives. CRE uses the NFR framework [21] to model NFRs.

In 2000, Ochs et al. [22] proposed the COTS acquisition process (CAP) which emphasized the concept of a "tailorable evaluation process". This means the evaluation process (including the evaluation criteria themselves) should be tailored based on the available effort for the project. Ochs' approach relied on experts' knowledge to tailor the process.

In 2001, a project was started by Chung et al. to define a COTS-Aware Requirements Engineering (CARE) approach [23-26]. CARE uses a flexible set of requirements based on different agents' views. For this, CARE proposes a method to define relevant agents as well as the system goals and requirements.

In 2002, another set of approaches were proposed:
(i) the PECA (Plan, Establish, Collect, and Analyze) approach [27] from SEI [5] describes a detailed tailorable COTS selection process and gives guidelines which the experts can use to tailor the process.
(ii) The BAREMO approach explained in detail how a decision can be made based on the AHP [7] method.
(iii) The storyboard approach [28] suggests to incorporate use-cases and screen-captures during the requirements engineering phase to help customers understand their requirements, and thus acquire more appropriate COTS products.
(iv) The combined selection approach [29] tries to select multiple COTS that are evaluated, first on the local scale to evaluate each COTS in isolation from the others, and then on the global scale to select the best combination of COTS.

In 2003, two more approaches were proposed:
(i) The WinWin spiral model [3] which is a risk-driven approach that suggests to identify, analyze and resolve risks in an iterative evaluation process.
(ii) The approach by Erol et. Al. [30] suggests the use of fuzzy theory to quantify qualitative data, and then to use optimization techniques to determine optimal (or near optimal) solutions.

In 2004, the DesCOTS system [31] was presented based on the work done in [11]. This system integrates several tools to define evaluation criteria using quality models such as ISO/IEC9126 [13].

In 2005, the MiHOS (Mismatch-Handling aware COTS Selection) approach was developed [32]. MiHOS relies on the GCS method and introduces a process for handling mismatches between COTS products and requirements. MiHOS uses techniques such as linear programming to identify near optimal solutions.

3.2. Surveying COTS selection approaches

Appendix 1 describes the eighteen approaches mentioned in section 3.1 in detail. Table 1 compares these approaches in terms of the following criteria:

1) GCS: Conformance to the *GCS* method.
2) EVAL: Evaluation strategy used.
3) SNG: Suitability for single COTS selection.
4) MLT: Suitability for multiple COTS selection.
5) MISM: Ability to address COTS mismatches in a systematic way during/after the selection process.
6) TAILOR: Tailorability of the process based on experts' knowledge. Satisfying this criterion does not necessarily imply the existence of any systematic tailoring techniques.
7) TS: Availability of tool support to facilitate the application of the approach.

Table 1 Comparing COTS selection approaches

APPROACH			COMPARISON FACTORS						
ID	Name	Year	GCS	EVAL	SNG	MLT	MISM	TAILOR	TS
A1	OTSO	95/96	✓	PF	✓	×	×	×	×
A2	IusWare	1997	✓	Any	✓	×	×	×	×
A3	PRISM	1997	✓	PF	✓	×	×	×	×
A4	CISD	1997	✓	PF/PZ	✓	~	×	×	×
A5	PORE	1998	✓	PF	✓	×	~	×	✓
A6	CEP	1999	✓	PF	✓	×	×	×	×
A7	STACE	1999	✓	KS/PF	✓	×	~	×	×
A8	CRE	1999	✓	PF	✓	×	~	×	×
A9	CAP	2000	✓	PF	✓	×	×	✓	×
A10	CARE	2001	✓	PF	✓	×	~	×	✓
A11	PECA	2002	✓	Any	✓	×	~	✓	×
A12	BAREMO	2002	step5	-	✓	×	×	×	✓
A13	Storyboard	2002	×	KS/PF	~	✓	~	×	×
A14	CS	2002	✓*	-	~	✓	~	×	×
A15	WinWin	2003	✓	PF	✓	~	~	×	×
A16	Erol's	2003	✓	-	✓	×	×	×	×
A17	DesCOTS	2004	✓	PF	✓	×	×	×	✓
A18	MiHOS	2005	✓	KS/PF	✓	×	✓	×	✓

PF: *Progressive filtering* — ✓ *fully satisfies the criterion*
KS: *Keystone* — × *does not satisfy the criterion*
PZ: *Puzzle assembly* — ~ *partially,informally,or implicitly satisfies the criterion*

**CS uses other approaches, e.g. OTSO, for local COTS evaluation.*

4. Evaluating existing approaches

In this section, we list some of the open research issues that have not been sufficiently addressed by existing COTS selection approaches.

As we have seen in section 3, there is a great variety of approaches to tackle the COTS selection problem. Practitioners have two main requests: (i) they are wondering about the effectiveness of the different approaches, and (ii) they need guidance on when and how to choose from these approaches. In his paper [1], R. Glass stated a clear message from software practitioners to software researchers: "*What help do practitioners need? We need some better advice on how and when to use methodologies*". This brings us to the first open research issue:

$\mathfrak{R}_1$: "*To support selection of appropriate and effective methods for our context*"

Furthermore, although most of the proposed approaches were developed for general use, there is no commonly accepted approach for COTS selection [6]. Also, these approaches were proposed without a clear explanation of how they can be adapted to different domains and projects. This means, even if we can satisfy $\mathfrak{R}_1$ and identify an approach that best-fits a specific context, customizing this approach to better fit the context would be problematic. This leads to the second open research issue:

$\mathfrak{R}_2$: "*To show how COTS selection approaches can be adapted to fit into different contexts*"

The next open issue is more relevant to the decision making during the selection process. Current approaches suggest using decision making techniques such as weighted score method (WSM) or analytic hierarchy process (AHP) [7]. However, there are several limitations to these techniques [33]. For example, these techniques estimate the fitness of COTS candidates based on 'one' total fitness score. This is sometimes misleading due to the fact that high performance in one COTS aspect might hide poor performance in another aspect. This leads to the third open research issue:

$\mathfrak{R}_3$: "*To use more informative decision making techniques*"

A follow up on the above problem is that, current decision making techniques usually suggest 'one' optimal solution to the decision makers who can either accept or reject it. This limits the decision makers' participation in and control over the decision making process. An alternative is to use decision support techniques. Decision support is a way of helping decision makers under different levels of uncertainty, complexity, and changing problem parameters [2]. Decision support suggests that instead of giving only one optimal solution, decision makers should be given several optimal or near optimal solutions from which they can either accept one or re-run the process again after adjusting its parameters until they reach a reasonable solution. This allows decision makers to have more participation in and control over the decision making process. This leads to the fourth open research issue:

$\mathfrak{R}_4$: "*To use decision support techniques in order to help making more efficient decisions*"

The next research issue is related to having different stakeholders with different views. Currently, this issue is resolved implicitly or informally. However, what is needed is a more robust negotiation component through which COTS can be progressively selected based on functional and non-functional requirements, architecture, and at the same time resolving conflicts between stakeholders. This leads to the fifth issue:

$\mathfrak{R}_5$: "*To provide a negotiation component for resolving conflicts between stakeholders*"

The remaining issues are related to the COTS market and their vendors. Most current approaches assume that the knowledge required for evaluating COTS products is available and reliable. Current approaches (e.g. [4]) suggest that COTS evaluation should initially rely on vendors' documentations. Then, for specific requirements, we should rely on vendor-led demonstration and on user-led experimentation. Nevertheless, the problem here is three-fold: (i) searching for and collecting the knowledge might very effort consuming. This leads to another open research issue:

$\mathfrak{R}_6$: "*To design knowledge repositories and show how they can be used for evaluating COTS*".

(ii) Some vendors are more reliable than others, and hence their documents can be used during the evaluation more broadly in order to reduce the evaluation effort related to user-led experimentations. This leads to another open research issue:

$\mathfrak{R}_7$: "*To develop techniques to evaluate vendors' credibility, and adjust the COTS evaluation results (obtained from vendors' documentations) based on such credibility estimation*".

5. An outlook

The seven issues described in section 4 can be analyzed for improving COTS selection practices. In this section, we show possible directions that can be used to address these eight issues.

For $\mathfrak{R}_1$: A high-level roadmap was given in [1] to support selecting appropriate methods for different contexts. The main idea is to taxonomize existing methods, to categorize problem domains in terms of what they need, and finally to find a way to map these two categories to each other. In addition, further confirmative studies should be conducted to analyze the effectiveness of the wide variety of approaches.

For $\mathfrak{R}_2$: Initial efforts [34, 35] developed a framework for customizing the COTS selection process. The main idea is to define a set of possible options for each activity during the process, and then give support to select the best option based on such project characteristics as effort, criticality, etc. Another useful reference can be found in [36]. Although this reference is not COTS oriented, it shows one method of process tailoring.

For $\mathfrak{R}_3$: The evaluation criteria can be defined in a hierarchical manner using goal-oriented definition [37]. Some COTS selection approaches already use goals [31]. However, it is not clear yet how to estimate the satisfaction of each goal. For this purpose, the relationships between goals should be quantified in terms of how each goal contributes to or prevents the satisfaction of other goals. The work done in [38] can be used as a reference for this purpose.

For $\mathfrak{R}_4$: There are many possible sources that explain the decision support concept [39-41]. An initial effort was done in [32] to incorporate decision support techniques in the COTS selection process.

For $\mathfrak{R}_5$: Negotiation components are essential parts of decision support systems [6]. Group decision making techniques [43, 44] might be very useful here. Also, agent-based systems can be employed to support the same purpose [45].

For $\mathfrak{R}_6$: The concept of learning software organization (LSO) [39] can be useful to help designing the required knowledge repositories. An initial effort was proposed in [46] where a framework for evaluating COTS with the support of knowledge bases is described. Also, there are useful online repositories for certain domains, e.g. [47]. However, more repositories are needed to cover the large spectrum of COTS domains.

For $\mathfrak{R}_7$: The work proposed by the approach A6 [48] (see Appendix-1) shows how to adjust evaluation results based on the credibility of the data source. However, different vendors should have different credibility factors which could be estimated using technique that can handle uncertain qualitative information, e.g. Bayesian Networks [49].

6. Conclusions

In this paper, we explored the evolution of COTS selection practices, and we surveyed and compared 18 of the most significant COTS selection approaches. In spite of the great variety of these approaches, there still many open issues related to the COTS selection process that need further research. We have listed some of these issues and showed some possible venues that could be taken to address them.

7. References

[1] D. G. Firesmith, "Achieving Quality Requirements with Reused Software Components: Challenges to Successful Reuse," in *MPEC'05*, St. Louis, Missouri, 2005.

[2] M. R. Vigder, W. M. Gentleman, & J. Dean, "COTS Software Integration: State of the art," NRC, 39198, Jan 1996.

[3] B.Boehm, D.Port, and Y.Yang,"WinWin Spiral Approach to Developing COTS-Based Applications,"*EDSER-5*,Oregon,2003.

[4] N.A.Maiden & C.Ncube, "Acquiring COTS Software Selection Requirements,"*IEEE Software,* vol.15(2),1998.pp. 46-56.

[5] SEI: Software Engineering Institute in Carnegie Mellon Univ, at http://www.sei.cmu.edu.

[6] G. Ruhe, "Intelligent Support for Selection of COTS Products,", *LNCS, Springer,* vol. 2593 2003. pp. 34-45

[7] T. L. Saaty, *The analytic hierarchy process*. New York: McGraw-Hill, 1990.

[8] D. Carney, "Evaluation of COTS Products: Some Thoughts on the Process," SEI Interactive, CMU university, Sept 1998,

[9] D. Carney, "COTS Evaluation in the Real World," SEI Interactive, Carnegie Mellon University, Dec 1998,

[10] J. Kontio, G. Caldiera, and V. R. Basili, "Defining factors, goals and criteria for reusable component evaluation," in *CASCON'96* Toronto, Ontario, Canada: IBM Press, 1996.

[11] X.Franch & J.P.Carvallo, "Using Quality Models in Software Package Selection,"*Software,*vol.20(1),Jan/Feb 2003. pp. 34-41

[12] J.P.Carvallo,X.Franch,G.Grau,&C.Quer,"COSTUME: a method for building quality models for composite COTS-based software systems," in *QSIC'04*, Germany, 2004, pp. 214 - 221

[13] ISO/IEC1926-1,*Software Engineering - Product Quality Model -Part1:Quality Model*. Geneve, Switzerland: ISO, 2001.

[14] D. Kunda and L. Brooks, "Identifying and Classifying Processes (traditional and soft factors) that Support COTS Component Selection: A Case Study," *ECIS'00*, Austria, 2000.

[15] P. A. Oberndorf, L. Brownsword, E. Morris, and C. Sledge, "Workshop on COTS-Based Systems," SEI Institute, CMU, Special Report CMU/SEI-97-SR-019, Nov. 1997,

[16] J. Kontio, "OTSO: A Systematic Process for Reusable Software Component Selection," University of Maryland, Maryland, CS-TR-3478, December 1995,

[17] M. Morisio and A. Tsoukis, "IusWare: a methodology for the evaluation and selection of software products," *IEE Software Engineering* vol. 144 (3), June 1997.

[18] R. W. Lichota, R. L. Vesprini, and B. Swanson, "PRISM: Product Examination Process for Component Based Development," *SAST '97*, 1997, pp. 61-69.

[19] V.Tran & D.Liu,"A Procurement-centric Model for Engineering Component-based Software Systems," *SAST '97*, 1997,pp.7079.

[20] D. Kunda & L.Brooks, "Applying social technical approach for COTS selection,"*UKAIS'99,* Univ of York, McGraw Hill, 1999.

[21] L.Chung, B.A.Nixon, E.Yu, and J.Mylopoulos, *Non-Functional Requirements in Software Engineering*: Kluwer Academic, 99.

[22] M.Ochs, D.Pfahl, G.Chrobok-Diening, and B. Nothhelfer-Kolb, "A COTS Acquisition Process: Definition and Application Experience," *ESCOM-SCOPE'00*, Munich, Germany, 2000.

[23] L. Chung and K. Cooper, "A COTS-Aware Requirements Engineering (CARE) Process: Defining System Level Agents, Goals, and Requirements," Department of Computer Science, The University of Texas, Dallas, TR UTDCS-23-01, Dec 2001.

[24] L. Chung and K. Cooper, "Knowledge-based COTS-aware requirements engineering approach,"*SEKE'02,*Ischia,Italy, 2002.

[25] L. Chung and K. Cooper, "Defining Goals in a COTS-Aware Requirements Engineering Approach," *Systems Engineering journal,* vol. 7 (1), 2004, pp. 61-83.

[26] L. Chung and K. Cooper, "Matching, Ranking, and Selecting COTS Components: A COTS-Aware Requirements Engineering Approach," in *MPEC'04,* Scotland, UK, 2004.

[27] S. Comella-Dorda, J. C. Dean, E. Morris, and P. Oberndorf, "A Process for COTS Software Product Evaluation," in *ICCBSS'02*, Orlando, Florida, 2002, pp. 86 - 96.

[28] S.Gregor, J.Hutson, & C.Oresky, "Storyboard Process to Assist in Requirements Verification and Adaptation to Capabilities Inherent in COTS," in *ICCBSS'02*, Florida, 2002, pp. 132-141.

[29] X. Burgues, C. Estay, X. Franch, J. A. Pastor, and C. Quer, "Combined Selection of COTS Components," in *ICCBSS'02*, Orlando, Florida, 2002, pp. 54-64.

[30] I. Erol and W. G. Ferrell-Jr., "A methodology for selection problems with multiple, conflicting objectives and both qualitative and quantitative criteria " *International Journal of Production Economics* vol. 86 (3), Dec 2003. pp. 187-199

[31] G. Grau, J. P. Carvallo, X. Franch, and C. Quer, "DesCOTS: A Software System for Selecting COTS Components," in *EUROMICRO'04*, Rennes, France, 2004, pp. 118-126.

[32] A. Mohamed, G. Ruhe, and A. Eberlein, "Decision Support for Handling Mismatches between COTS Products and System Requirements," in *ICCBSS'07,* Banff, Canada, 2007.

[33] C. Ncube and J. C. Dean, "The Limitations of Current Decision-Making Techniques in the Procurement of COTS Software Components," in *ICCBSS 2002*, 2002, pp. 176-187.

[34] A. Mohamed, G. Ruhe, and A. Eberlein, "Towards a Customizable Approach for COTS Selection," in *SEA2004*, Cambridge, MA, USA, 2004, pp. 665-671.

[35] A. Mohamed, G. Ruhe, and A. Eberlein, "Customization of COTS selection Process," SEDS Lab, UofC, Calgary, TR042/2005, *http://www.enel.ucalgary.ca/~asamoham/SEDS/TR042-2005-Short.pdf.*

[36] L. Jiang, A. Eberlein, and B. H. Far, "Methodology for Requirements Engineering Process Development," in *ECBS'04* Brno, Czech Republic, 2004.

[37] A. V. Lamsweerde, "Goal-Oriented Requirements Engineering: A Guided Tour," in *5th IEEE International Symposium on Requirements Engineering*, 2001, pp. 249-263.

[38] H. Kaiya, H. Horai, and M. Saeki, "AGORA: attributed goal-oriented requirements analysis method," in *RE'02*, Essen. Germany, 2002, pp. 13- 22.

[39] G. Ruhe, "Software Engineering Decision Support - A New Paradigm for Learning Software Organizations," *Lecture Notes in Computer Science,* vol. 2640 Nov 2003. pp. 104-113
[40] G. Ruhe, A. Eberlein, and D. Pfahl, "Quantitative WinWin: a new method for decision support in requirements negotiation," in *SEKE'02*, Ischia, Italy, 2002, pp. 159-166.
[41] G. Ruhe and A. Ngo-The, "Hybrid Intelligence in Software Release Planning," *International Journal of Hybrid Intelligent Systems,* vol. 1 (2), 2004. pp. 99-110
[43] C. L. Hwang and M. J. Lin, *Group Decision Making under Multiple Criteria-Methods and Applications* Springer-Verlag, Berlin, Heidelberg, New York, NY, 1987.
[44] F. Herrera, E. Herrera-Viedma, and J. Verdegay, "A Model of Consensus in Group Decision Making under Linguistic Assessments," *Fuzzy Sets and Systems,* vol. 78 1996. pp. 73-87
[45] T. Wanyama and B. H. Far, "A Multi-Agent Framework for Conflict Analysis and Negotiation: Case of COTS selection," *Transactions of Institute of Electronics, Information and Communication IEICE, Special Issue of Software Agent and its Applications,* vol. E88-D (9), Sept. 2005. pp. 2047-2058
[46] A. Mohamed, T. Wanyama, G. Ruhe, A. Eberlein, and B. Far, "COTS Evaluation Supported by Knowledge Bases," *Lecture Notes in Computer Science,* vol. 3096 2004/09 2004. pp. 43-54
[47] CMS-Matrix, "Avail. at http://www.cmsmatrix.org."
[48] B. C. Phillips and S. M. Polen, "Add Decision Analysis to Your COTS Selection Process," *The Journal of Defense Software Engineering,* April 2002 2002.
[49] F. V. Jensen, *Introduction to Bayesian Networks*. NewYork: Springer-Verlag, 1996.
[50] J. Kontio, "A Case Study in Applying a Systematic Method for COTS Selection," in *18th International Conference on Software Engineering*, Berlin, Germany, 1996, pp. 201-209.
[51] J. Kontio and R. Tesoriero, "A COTS selection method and experiences of its use," in *The Twentieth Annual Software Engineering Workshop* Greenbelt, Maryland, 1995.
[52] C. Ncube and N. Maiden, "Guiding parallel requirements acquisition and COTS software selection," in *IEEE International Symposium on Requirements Engineering*, University of Limerick, IRELAND, 1999, pp. 133 - 140.
[53] C. Ncube and N. A. Maiden, "PORE: Procurement-Oriented Requirements Engineering Method for the Component Based Systems Engineering Development Paradigm," *Intr. Workshop on Component Based Software Eng.,* Los Angeles, CA, 1999.
[54] B. Kitchenham, "DESMET: A method for evaluating Software Engineering methods and tools," Department of Computer Science, University of Keele, UK, TR96-09, 1996,
[55] D. Kunda, "A social-technical approach to selecting software supporting COTS-Based Systems," PhD Thesis, Department of Computer Science, University of York, Oct 2001
[56] S. Kunda, "A social-technical approach to selecting software supporting COTS-based systems," Thesis, Univ. of York, 2001
[57] C. Alves and J. Castro, "CRE: A Systematic Method for COTS Components Selection," *SBES'01* Rio de Janeiro, Brazil, 2001.
[58] S. Comella-Dorda, J. C. Dean, G. Lewis, E. Morris, P. Oberndorf, and E. Harper, "A Process for COTS Software Product Evaluation," CMU Univ., SEI Institute, CMU/SEI-2003-TR-017, July 2004.
[59] A. Lozano-Tello and A. Gomez-Pérez, "BAREMO: how to choose the appropriate software component using the analytic hierarchy process," in *SEKE'02*, Italy, 2002, pp. 781-788.
[60] ExpertChoice-homepage:: http://www.expertchoice.com.
[61] Y. Yang, J. Bhuta, B. Boehm, D. Port, and C. Abts, "Composable Process Elements for Developing COTS-Based Applications," in *EDSER-5*, Portland, Oregon, 2003.
[62] B. Boehm, "A Spiral Model of Software Development and Enhancement," *Computer,* vol. 21 May 1988. pp. 61-72
[63] Y. Akao, *Quality Function Deployment: Integrating Customer Requirements into Product Design*. Cambridge, MA: Productivity Press, 1999.

Appendix-1: Current COTS Selection Approaches in a Nutshell

This appendix summarizes current COTS selection approaches. The approaches are described in tables 2 to 19. For each approach, the following items are described:

- The main idea that distinguishes this approach from other approaches.
- The main steps of the approach.
- The evaluation strategy used by the approach.
- The pros and cons of the approach.
- The availability of tool support (TS).

Please refer to sections 2.1and 2.2 for further details about these items.

Many of the approaches described in this appendix share the following cons:

- *REQ-ASSUMPTION*: the approach assumes the requirements already exist and fixed.
- Multi-COTS: the approach does not support multiple COTS selection for COTS intensive systems.
- *MISMATCHES*: It is not clear how to deal with COTS mismatches. The approach assumes that there is a set of COTS products that satisfy most of the requirements, at least to an acceptable level.
- *AHP/WSM-WEAKNESSES*: the approach uses AHP or WSM, and therefore inherits their weaknesses; e.g. consolidating the results into a single score is misleading, because strong aspects mask weak ones.

The acronyms described above are used when describing the approaches.

Table 2 The OTSO Approach

A1 : OTSO Approach (1995) [10, 16, 50, 51]	
Main Idea	OTSO (Off-The-Shelf Option) compares COTS products based on two factors: *value* and *cost*: (i) The *value* is estimated based on hierarchical evaluation criteria which consist of functionalities, qualities, strategic concerns, and architectural constraints. The evaluation criteria is influenced by 5 factors: application requirements and architecture, project constraints, availability of required features, and organization infrastructure (e.g. level of experience). AHP or WSM [7] are used to calculate the overall value of each alternative. (ii) The *cost* is estimated based on: acquisition cost, further development costs, and integration cost. The cost and benefit of each alternative are consolidated using AHP[7]
Main Steps	1. *Evaluation criteria*: define evaluation criteria. 2. *Search*: search the market for possible COTS. 3. *Screening*: filter out the COTS that do not comply with the must-have requirements. 4. *Evaluation*: evaluate the benefit and cost of COTS. 5. *Analysis*: use AHP/WSM to consolidate the evaluation results and select a COTS. 6. *Deployment*: integrate the selected COTS into system. 7. *Assessment*: assess the success of the selection process as a feedback for future uses.
Eval. Str.	▪ Progressive filtering
Pros	▪ Since OTSO is one of the first approaches reported in the literature, it served as a basis for other approaches. ▪ The feedback from the 'assessment' step helps to improve future selection processes in the organization.
Criticism	▪ *REQ-ASSUMPTION, Multi-COTS, MISMATCHES, AHP/WSM-WEAKNESSES*
TS	▪ Not available.

Table 3 The IusWare Approach

A2 : IusWare Approach (1997) [17]	
Main Idea	IusWare (IUStitia softWARis) is a two phase approach that is designed to evaluate COTS products in a formal and rigorous way. IusWare relies on multi-criteria decision aid (MCDA) [33] to select COTS products.
Main Steps	IusWare is a two phase approach: Phase 1. Design the evaluation model a. Identify the actors relevant to the evaluation, their role, objective of the evaluation, and available resources.

	b. Identify the evaluation type, either (i) ranking products from highest to lowest, or classifying products in 2 sets:the best and the remaining. (ii) formal description of the products. c. Define a hierarchy of evaluation attributes, often corresponding to quality attributes. d. Define the measures for the evaluation attributes e. Choose an aggregation technique, e.g. WSM or AHP[7] Phase 2. Apply the evaluation model
Eval. Str.	▪ Depend on Step (b) in Phase1.
Pros	▪ Focus on quality attributes. ▪ Formalize the evaluation process (although this might be seen sometimes as a disadvantage with light-weight projects -as it increases the complexity of the process).
Criticism	▪ *REQ-ASSUMPTION, Multi-COTS, MISMATCHES*
TS	▪ Not available.

Table 4 The PRISM Approach

A3 : PRISM Approach (1997) [18]	
Main Idea	The PRISM (Portable, Reusable, Integrated, Software Modules) approach comprises two parts: (i) a generic component architecture, and (ii) a product evaluation process (PEP). The PEP process focuses on prototyping to ensure the selected product complies with industry standards represented by the generic architecture.
Main Steps	1. Identification: based on initial criteria, identify products that fit into the generic architecture. 2. Screening: identify one or more products with best fitness for further examination. 3. Stand-alone test: evaluate products against reliability, reusability, and system requirements. 4. Integration test: estimate how readily the product is to be integrated to the architecture. 5. Field test, re-evaluate the product after deployment in the target context
Eval. Str.	▪ Progressive filtering
Pros	▪ Address make-or-buy decisions. ▪ Address issues related to the integration into system. ▪ Emphasize the importance of prototyping: first outside the context, and second inside the context.
Criticism	▪ Identifying a generic architecture can be in many cases valid only for specific context. ▪ Many aspects of the approach are vague, e.g. how to acquire the requirements?, how to consolidate the results?, how to define the test cases?, etc. ▪ *REQ-ASSUMPTION, Multi-COTS, MISMATCHES.*
TS	▪ Not available.

Table 5 The CISD Approach

A4 : CISD Approach (1997) [19]	
Main Idea	*The CISD (COTS-based Integrated Systems Development)* is a waterfall-style approach to identify and evaluate COTS products and then integrate them into the system.
Main Steps	a) Product identification a1. Identify requirements a2. Identify products and collect data about them a3. Prioritize products into sets for further evaluation. b) Product evaluation: Evaluate the product sets to find the optimal combination for integration. The optimal combination should include a set of collaborative COTS products. The products are evaluated for their 'individual functionality', 'interoperability of products', and 'performance of individual products and product sets'. c) Product integration: Build necessary product adapters into the selected product set.
Eval. Str.	▪ Progressive filtering + Puzzle assembly
Pros	▪ Support the selection of multiple COTS components in COTS intensive systems
Criticism	▪ CISD is a waterfall-style approach as each phase depends on the previous one ▪ Not clear how to evaluate a selected set of products and how they interact, e.g. in terms of resources. ▪ *REQ-ASSUMPTION, MISMATCHES*
TS	▪ Not available.

Table 6 The PORE Approach

A5 : PORE Approach (1998) [4, 52, 53]	
Main Idea	The PORE (Procurement-Oriented Requirements Engineering) approach focuses on the requirements engineering phase of the COTS procurement process. It suggests acquiring the requirements as the same time as acquiring and analyzing COTS products. PORE suggests iterating between requirements acquisition and product selection and rejection until finding a COTS that satisfies a sufficient number of the requirements (i.e. progressive filtering evaluation). PORE integrates techniques such as: feature analysis techniques [54] to help scoring the compliance of COTS to requirements, and multi-criteria decision making techniques [7] to help selecting a COTS.
Main Steps	PORE defines three templates for evaluating COTS products through three main phases: *Phase1.* First template gives guidance to acquire customer requirements and product information from COTS vendors given documents and information. *Phase2.* Second template gives guidance to acquire customer requirements and product information from vendors-led demonstrations; using test cases. *Phase3.* Third template gives guidance to acquire customer requirements and product information from customer-led product exploration; using trial versions
Eval. Str.	▪ Progressive filtering
Pros	▪ The parallel requirement acquisition and COTS selection means the defined requirements inform the selection process and vice versa, which is more realistic than assuming a fixed set of requirements. ▪ PORE suggests using the Analytic Hierarchy Process (AHP) method [7] only after Phase3 after defining a small number of products. this leads to minimizing the extra effort that is required by the AHP method.
Criticism	▪ PORE (as stated by its authors [52]) is labor intensive. ▪ PORE partially addresses the COTS mismatch problem. The progressive definition of the system requirements allow to refine those requirements in a way that reduces the mismatches with the products specifications. However, it is not clear it is till not clear how to deal with the situation when many mismatches are still unresolved. PORE does not define a systematic process that can be applied to (i) identify the influence of the remaining mismatches on the COTS selection decision, and (ii) to handle the remaining mismatches after selection. ▪ PORE does not provide clear guidelines on when to stop acquiring requirements and COTS information. ▪ *Multi-COTS, AHP/WSM-WEAKNESSES.*
TS	▪ PORE Process Advisor (prototype tool)

Table 7 The CEP Approach

A6 : CEP Approach (1999) [48]	
Main Idea	The CEP (Comparative Evaluation Process) approach is an advancement of OTSO approach. CEP suggests the use of the credibility feature. The more confidence in the source of data is, the higher 'confidence factor (CF)' that source gets. The CF is multiplied by the criteria scores when making product selection decisions. For example, consider a criterion is evaluated and gets a score of 100% satisfaction by a COTS product. If this criterion is evaluated based on 'verifying' the criterion, then CF=1.0 and the final score is still 100%. However, if the evaluation is based on 'vendor documents', then CF = 0.3 and the final score is 30% only. Eventually, CEP uses weighted averages to consolidate the evaluation results.
Main Steps	1. *Scoping evaluation effort*: set the expected effort and schedule for the evaluation activities. 2. *Searching / Screening*: search for COTS candidates

	based on 'must-have' criteria, and filter out the ones that do not satisfy them. 3. *Criteria definition*: give a detailed definition of the evaluation criteria. CEP defines several categories of evaluation criteria: functional, basic, architecture, management, and strategic. 4. *Evaluation*: evaluate COTS alternatives and estimate the CF factor for each criterion. 5. *Analysis*: analyze evaluation results and compare COTS alternatives.
Eval. Str.	▪ Progressive filtering
Pros	▪ Relies on a detailed evaluation criteria (i.e. related to management, etc – see main steps) ▪ Introduce the CF concept (although it might still need some improvements to be applicable)
Criticism	▪ Using the same CF value for all vendors is a strong assumption. Some vendors are more reliable and credible than others. ▪ *REQ-ASSUMPTION, Multi-COTS, MISMATCHES, AHP/WSM-WEAKNESSES*
TS	▪ Not available.

Table 8 The STACE Approach

A7 : STACE Approach (1999) [14, 20, 55, 56]	
Main Idea	STACE (Social-Technical Approach to COTS Evaluation) emphasizes the importance of non-technical issues when defining the evaluation criteria and conducting the evaluation process. The non-technical issues include social, human, and organizational characteristics, e.g. political and economic factors. STACE also emphasizes the customer participation during evaluation. In [55], studies were conducted to compare the application of STACE in developing and developed countries.
Main Steps	1. *Requirements elicitation*: the high level requirements are elicited from stakeholders, market studies, system documents, and domain knowledge 2. *Social-technical criteria definition*: the high level requirements are decomposed into measurable attributes. The decomposition addresses the non-technical characteristics (e.g. social-economic). 3. *Alternatives identification*: the high level requirements are decomposed into measurable attributes. The decomposition addresses the non-technical characteristics (e.g. social-economic). 4. *Evaluation*: the COTS alternatives are evaluated and ranked according to the social-technical criteria.
Eval. Str.	▪ Keystone evaluation, and Progressive filtering
Pros	▪ Incorporate the non-technical characteristics in the evaluation process. ▪ Support the negotiation between requirements elicitation and COTS evaluation through emphasizing customer involvement.
Criticism	▪ STACE is a very extensive process that requires more effort to apply. ▪ *Multi-COTS, MISMATCHES, AHP/WSM-Weaknesses*.
TS	▪ Not available.

Table 9 The CRE Approach

A8 : CRE Approach (1999) [57]	
Main Idea	CRE (COTS-based Requirements Engineering) emphasizes the importance of non-functional requirement (NFR) as a decisive criteria when comparing COTS alternatives. CRE uses NFR framework [21] to model the NFRs for this purpose. Evaluating all NFRs requires more effort than most organizations have. So, CRE suggests select the most promising COTS candidates for detailed evaluation.
Main Steps	1. *Identification*: the core requirements and COTS candidates are identified. 2. *Description*: further requirements are defined. The NFR framework is used to model the NFRs. 3. *Evaluation*: COTS candidates are evaluated and WSM or AHP [7]are used for decision making. 4. *Acceptance*: the highest ranked COTS is selected if it passes some legal acceptance tests; if not, the next COTS is taken, and so on.
Eval. Str.	▪ Progressive filtering.
Pros	▪ Using the NFR framework during COTS evaluation facilitates addressing the NFRs.
Criticism	▪ Evaluating all NFRs (even for small set of COTS candidates) adds extra effort that is not available in most real situations. ▪ It is not clear how quality issues are verified, e.g. how a required level of quality is reached. ▪ It is not clear how to deal with unsatisfied quality attributes. ▪ *Multi-COTS, MISMATCHES, AHP/WSM-Weaknesses*.
TS	▪ Not available.

Table 10 The CAP Approach

A9 : CAP Approach (2000) a; Ochs, 2001 #184}}	
Main Idea	CAP (COTS Acquisition Process) is a measurement oriented approach where the evaluation process is tailored based on an estimation of the measurement effort. The tailoring is applied to: (i) a general evaluation criteria (called evaluation taxonomy) that is defined as a part of CAP, and (ii) the evaluation depth ranging from using only documents (as source of data) to using prototyping.
Main Steps	CAP compromises three main components: a) "Initialization", which deals with planning the evaluation process and its cost estimation: a1. Tailor and weight taxonomy of evaluation criteria a2. Estimate the cost of applying CAP a3. Elaborate measurement plane b) "Execution", which provides guidance to conduct the evaluation process itself: b1. *Exploration* for (i.e. identification of) COTS products b2. *Collecting measures*, where the COTS products are initially evaluated b3. *Screening*, where products with less compliance with the criteria are filtered out b4. *Collect measures*, where the COTS products are evaluated more extensively b5. *Ranking of COTS using AHP* [7] b6. *Make-or-Buy*, where the highest ranked COTS is selected if it passes a make-or-buy decision. c) "Reuse" component, where all info. gathered by other parts are stored for future use.
Eval. Str.	▪ Progressive filtering
Pros	▪ The introduction of the concept "tailorable evaluation process" which is useful to customize the process based on available resources. ▪ The introduction of a reusable taxonomy for the evaluation criteria.
Criticism	▪ *REQ-ASSUMPTION, Multi-COTS, MISMATCHES, AHP/WSM-WEAKNESSES*.
TS	▪ Not available.

Table 11 The CARE Approach

A10 : CARE Approach (2001 - ongoing) [23-26]	
Main Idea	CARE (COTS-Aware Requirements Engineering) approach focuses on the requirements engineering phase of the selection process. CARE defines two types of requirements: *native* (requirements acquired from customers) and *foreign* (requirements that may be implemented by existing COTS products). CARE tries to bridge the gap between the native and the foreign requirements by either asking the customers to change a goal or requirements or asking the vendors to customize the COTS product.
Main Steps	1. *Define goals*: define system goals (i.e. native goals) 2. *Match goals*: search for COTS products that match the goals (functional first, non-functional second).

	3. *Rank components*: perform a gap analysis to rank COTS products. 4. *Negotiate changes*: if mismatches found, negotiate changes to system goals or COTS components. 5. *Select Components*: select a set of COTS components.
Eval. Str.	▪ Progressive filtering
Pros	▪ Suggest to use flexible requirements which is more realistic than defining a fixed set of requirements ▪ Partially addresses the COTS mismatch problem. ▪ Support the definition of a plausible set of requirements based on different agents views.
Criticism	▪ Although CARE emphasizes the importance to map system requirements to product specs, the following weaknesses was identified: CARE suggests negotiating the resolution of the mismatches and how this influences the system and the product. However, this was given as a high level guideline which means, there is no systematic process that can be applied to (i) identify the influence of these mismatches on the COTS selection decision, and (ii) to handle the identified mismatches. What CARE suggests is similar to what currently happens (in an ad-hoc manner) in real-world selection. ▪ Not clear how to define requirements for multiple COTS selection in COTS intensive systems.
TS	▪ CARE Assistant Tool (prototype tool)

Table 12 The PECA Approach

A11 : PECA Approach (2002) [27, 58]	
Main Idea	PECA (Plan, Establish, Collect, and Analyze) is a high level process that has to be tailored for each specific project. PECA elaborates all activities in the evaluation process, and provides several alternatives for each activity to be used during process tailoring.
Main Steps	a) Plan the evaluation. a1. Form evaluation team a2. Create charter a3. Identify stakeholders a4. Pick the approach a5. Estimating resource & schedule b) Establish the criteria b1. Identify evaluation criteria b2. Construct criteria c) Collect the data - Estimate how each product complies with the evaluation criteria d) Analyze the data - analyze the data to select a COTS product using some techniques such as gap analysis.
Eval. Str.	▪ Depend on the project context.
Pros	▪ The tailorability of PECA makes it applicable in many contexts.
Criticism	▪ No clear guidance on how to tailor the process. the process only defines different alternatives, and the tailoring process based on human experience. ▪ *Multi-COTS, MISMATCHES.*
TS	▪ Not available.

Table 13 The BAREMO Approach

A12 : BAREMO Approach (2002) [59]	
Main Idea	The BAREMO (Balanced Reuse Model) focuses on how to use the AHP technique [7] to select COTS products.
Main Steps	1. Specify project objectives 2. Construct a decision tree (where root node is the problem objective, intermediate nodes are criteria, leaf nodes are alternatives) 3. Generate a comparison matrices for criteria at the same level in the decision tree 4. Value each alternative at the leaf nodes with respect to connected criteria at intermediate nodes. 5. Calculate a final value for each alternative using weighted addition scales.
Eval. Str.	▪ Not applicable.
Pros	▪ .Inherit the advantages of AHP technique.
Criticism	▪ Does not define a COTS evaluation / selection mechanism. ▪ Although BAREMO investigates the use of AHP in details, the concept itself is not new and has been already used in many other approaches.. ▪ *REQ-ASSUMPTION, Multi-COTS, MISMATCHES, AHP/WSM-WEAKNESSES* .
TS	▪ . Any available AHP tool, e.g. ExpertChoice [60]

Table 14 The Storyboard Approach

A13 : Storyboard Approach (2002) [28]	
Main Idea	Storyboard approach incorporates 'use cases' and 'screen captures' during the requirements engineering phase to help customers understand their requirements, and thus, acquire a more appropriate COTS that can be integrated with minimal customization.
Main Steps	1. Determine the requirements that can be satisfied by COTS products. 2. Develop use-cases for the identified requirements. 3. Identify and evaluate COTS products based on Steps 1 and 2 4. Based on the use-cases in Step2, develop storyboard using screen captures. Screen captures can be taken from actual COTS products, or from a custom interface developed using e.g. JAVA.
Eval. Str.	▪ Progressive filtering and Key-stoning.
Pros	▪ Provide a means to communicate requirements among stakeholders more clearly and concisely. ▪ Provide a means to manage users' expectations. ▪ Support the selection of multiple COTS components in COTS intensive systems.
Criticism	▪ Non functional requirements are not addressed. ▪ No formal evaluation process is presented. ▪ Not clear how to handle possible conflicts between the COTS products integrated together; i.e. resource usage. ▪ *MISMATCHES.*
TS	▪ Not available.

Table 15 The Combined-Selection Approach

A14 : Combined Selection Approach (2002) [29]	
Main Idea	The Combined-Selection (CS) approach is used to select multiple COTS products that all together satisfy the requirements. This approach performs its activities at two levels: local and global. *The global level* (i) addresses the overall process of the combined selection, (ii) fires individual selection processes for each area, and (iii) tries to find the best overall combination of products. *The local level* use existing COTS evaluation and selection techniques (e.g. OTSO [16] or PORE [4]) to select individual COTS that are combined at the global level.
Main Steps	1. Plan the global selection process and firing individual selection processes. (global level) 2. Identify COTS candidates for individual areas. (local level) 3. Identify global COTS integration scenarios; e.g. product A will cover area 1, while product B will cover area 2 and part of area 1. (global level) 4. Evaluate individual scenarios at each individual area. (local level) 5. Evaluate integration scenarios (global level). 6. Select the COTS products.
Eval. Str.	▪ Depends on the process used at the local level; e.g. progressive filtering, if PORE is used.
Pros	▪ Support the selection of multiple COTS components in COTS intensive systems.
Criticism	▪ CS inherits the weaknesses of the method used at the local level. ▪ No formal models are presented to guide the integration process and identifying the conflicts between the selected COTS components.

	▪ *REQ-ASSUMPTION, MISMATCHES.*
TS	▪ Not available.

Table 16 The WinWin Approach

A15 : WinWin Approach (2003) [3, 61]	
Main Idea	The WinWin spiral model is a risk-driven approach adapted from the classical software development spiral model [62]. WinWin follows an iterative process in which the requirements are acquired in parallel to evaluating the COTS products. WinWin emphasizes concurrent product identification and process refinement. A decision framework is used to provide guidance for the COTS-based development decisions, e.g. make-or-buy, COTS selection, COTS tailoring, glue-coding, etc.
Main Steps	WinWin uses five iterative steps, in which more stakeholders and OC&P (Objectives, Constraints and Priorities) are identified in successive iterations, and more refinement of the process is applied: 1. Identify stakeholders, and system OC&P 2. Evaluate products with respect to OC&P, and address risks (e.g. related to customer support) 3. Elaborate product and process definition 4. Verify and validate product and process definition. 5. Stakeholders' review and commitments
Eval. Str.	▪ Progressive filtering
Pros	▪ Addressing risks early in the selection process reduces the cost of handling them in the future. ▪ The decision framework provides an inclusive description of the actual development process. ▪ Gradually defining the stakeholders' needs as well as the process fits more in many real-world situations. ▪ WinWin addresses the selection of multiple COTS components in intensive COTS-based system.
Criticism	▪ Although multiple COTS selection is support, no detailed guidelines (or formal process) were defined to elaborate this issue. ▪ Although WinWin emphasizes the importance to address risks (including those related to COTS mismatches) during its spiral process, this was given as a set of high level guidelines. That is, there is no systematic process that can be applied to (i) identify the influence of these mismatches on the COTS selection decision, and (ii) to handle the identified mismatches after selection.
TS	▪ Not available.

Table 17 The Approach by Erol et al.

A16 : The Approach by Erol et al. (2003) [30]	
Main Idea	The approach by Erol et al. is an evaluation approach that supports selecting a COTS product from a finite set of products based on: (i) more than one objective and (ii) a set of quantitative (e.g. cost) and qualitative (e.g. linguistic variables) data. The approach uses fuzzy QFD (Quality Function Deployment) [63] to collect and quantify the qualitative data. Then, goal programming (a generalization of linear programming) is used to suggest near optimal solutions to the decision maker.
Main Steps	1. Acquire requirements and product information. 2. Transform qualitative data to quantitative data using fuzzy QFD. 3. Construct a multi-objective model for objectives: (i) maximize customer value, and (ii) minimize cost. 4. Solve the model several times with slightly different parameter values to get multiple feasible solutions from which the decision maker can choose one solution.
Eval. Str.	▪ Not applicable
Pros	▪ Address qualitative data. ▪ Provide more than one feasible solutions to support making the final decision.
Criticism	▪ Do not address activities other than COTS evaluation. ▪ *REQ-ASSUMPTION, Multi-COTS, MISMATCHES.*
TS	▪ Not available.

Table 18 The DesCOTS system

A17 : DesCOTS system [31]	
Main Idea	DesCOTS ((**D**escription, **e**valuation and **s**election of **COTS** components) system includes a set of tools that can be used to evaluated COTS products based on quality models such as ISO/IEC9126 [13]. DesCOTS is built on the work proposed in [11]
Main Steps	DesCOTS follows the GCS process and uses the following steps to define the evaluation criteria: 1. From ISO/ICE9126, determine the quality characteristics and subcharacteristics that will be used. 2. refine the subcharacteristics as need; e.g. 'suitability' to basic suitability, and added suitability 3. Refine subcharacteristics into measurable attributes. 4. Refine derived attributes to basic ones; e.g. GUI quality to user friendliness, depth of menus, etc. 5. Identify relationships between quality entites 6. Determine metrics for attributes.
Eval. Str.	▪ Progressive filtering.
Pros	▪ Detailed method to define the evaluation criteria. ▪ Considers quality characteristics. ▪ Can be adapted to many domains.
Criticism	▪ *Multi-COTS, MISMATCHES.*
TS	▪ Yes

Table 19 The MiHOS Approach

A18 : MiHOS Approach [32]	
Main Idea	MiHOS (Mismatch-Handling aware COTS Selection) focuses on handling the mismatches between COTS candidates and the requirements. MiHOS firstly provides guidelines to handle the mismatches that are detected during the COTS selection process, and secondly uses decision support techniques to (i) study the cost of resolving these mismatches on the COTS selection decision, and (ii) support handling the mismatches of the selected COTS product.
Main Steps	MiHOS has two components: the "COTS selection" component which follow exactly the GCS process, and the "mismatch handling" component which is divided into two parts: <u>Part1 (of mismatch handling component):</u> 1.1 Analyze the detected mismatch 1.2 Decide whether the mismatch should be tolerated or resolved by (i) adjusting the requirements, or by (ii) adjusting the COTS product. If (ii) is chosen, then the mismatch is postponed until a COTS is selected. <u>Part2 (of the mismatch handling component):</u> 2.1 Modeling: define problem parameters, e.g. available resources for resolving the mismatches of the selected COTS product, the possible techniques to resolve each mismatch, cost of each technique, etc. 2.2 Exploration: use optimization techniques to fine optimal or near optimal solutions. 2.3 Consolidation: experts should review the results and either accept one solution or adjust the problem parameters in (2.1) and run the model again.
Eval. Str.	▪ Keystone identification / Progressive filtering
Pros	▪ Can handle mismatches during/after selection process. ▪ Uses goal-driven criteria definition which allows for more informative decisions. ▪ Use interactive decision support techniques which enable decision makers to have more participation in and control over their decisions.
Criticism	▪ Requires effort more than most selection approaches. ▪ Is not intended to address quality requirements. ▪ No support for the selection of multiple COTS components in COTS intensive systems ▪ *Multi-COTS.*
TS	▪ MiHOS-TS, a prototype.

Modern Distributed Data Acquisition and Control Systems based on OPC Techniques

Vu Van Tan, Dae-Seung Yoo, and Myeong-Jae Yi
School of Computer Engineering and Information Technology,
University of Ulsan, San-29, Moogu-2 Dong, Namgu, Ulsan 680-749, Republic of Korea,
Email: {vvtan, ooseyds, ymj}@mail.ulsan.ac.kr

Abstract

Web technologies are gaining increased importance in automation control systems, especially in slow control systems. Alternatively, the requirements to exchange the complex data between the cooperating applications and the applications-based non-Microsoft Platforms are widely increasing in the industrial systems. The applications-based OPC (OLE for Process Control) technique to exchange data between the measurement and control systems on plant floor with XML leverages are proposed in the OPC XML DA (Data Access) Specification. Moreover, we are expecting a system that allows easily aggregating the existing productions and maintenance managements with new products as frameworks. This system also allows exchanging the data between collaborating applications in horizon. Besides, the high performance and the ability to read and decode any type of the data from measurement and control systems are important factors in the application design. This paper introduces new control system aspects that are used for the design of slow control systems. In general, they are based on the OPC and XML technologies. Main aspects are the throughout of standards for interfaces, functionalities and architectures. To guarantee the security problems in system, this paper also discusses the levels of the security in automation and control systems. These discussions provide more security solutions for technical-level readers.

Keywords: OPC, data acquisition, slow control system, XML, control system aspects, security

1. Introduction

Web technologies have been adopted not only in the e-business but also in the automation and control systems, especially in slow control systems. In addition, XML is an open standard that provides interoperability and data integration using Internet. It also is the preferred format for encoding and moving structured data in the independent system. XML and Internet technologies provide new and powerful ways to assessing and delivering plant floor, condition monitoring, e-diagnostic to users across manufacturing enterprise using standard Web-browsers and wireless devices [1]. Therefore, OPC Foundation formed the OPC XML-DA technical working group to define a new specification to move the same type of plant floor data as the existing OPC COM-DA. The OPC XML-DA provides vertical integration between the plant floor and condition monitoring system, maintenance system, and enterprise application using such as industrial standards, XML, and SOAP (Simple Object Access Protocol) [2], [3]. Additionally, OPC Foundation has defined the OPC Complex Data for implementing both the OPC DA and XML-DA. The OPC Complex Data [4] working group is making enhancements to OPC specifications based on requirements and feedback from other industry groups to address additional data types such as structures, binary, XML, etc. The design and implementation of the XML-DA Server that allows OPC Clients to read and decode any type of data from hardware I/O devices (i.e. measurement and control systems) are represented in [2]. Another aspect mentioned in [5], the authors proposed a data exchange protocol based on OPC XML-DA. This protocol has focused on the acceptable performance by providing several extensions. It was interested in the consistency with high level standards, multi-platform compatibility, and high performance.

However, the requirements to convert data between XML representation and OPC binary representation make it to determine the required size of the memory buffer to hold the whole data. Moreover, the only fast possibility to transmit large amount of data is to use binary representation rather than XML representation. In addition, different platforms even different compilers of the same platform use different binary

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

representations such as different floating point formats, different string formats, etc [4], [6]. Therefore, some universal standards should be chosen to transport data between different platforms. Besides these problems, the OPC XML-DA has a big disadvantage that uses XML textual data representation. This causes much more network traffic to transfer data [6]. Thus, the OPC XML-DA improvements that are to reach an acceptable system performance are required. Moreover, with XML-DA leverages, it requires to aggregate the thousands of existing OPC COM-DA products use in today into a system that supports to exchange data between OPC Servers. This system should be configured by using remote clients through the Internet.

This paper proposes new control system aspects for slow control systems that are based on OPC and XML technologies. These design aspects include interfaces, functionalities, and architectures. This system allows aggregating the thousands of existing OPC COM-DA products use in today with XML-DA Servers into a system for supporting the horizontal data exchange between them. The security aspects are also discussed.

This paper is organized as following sections: The next section shows the model of Web integration, overviews of related OPC techniques, and the problem statements. Section 3 proposes the framework for design and implementation of the Universal DA Server using in slow automation and control systems. Section 4 investigates the database connection and data representation between Universal DA Server and OPC Clients with results of benchmarks. In Section 5, the security aspects are discussed. Finally, we mark some summaries and future works in Section 6.

2. Background and Problem Statements

Continuously, we show the model of web integration in Section 2.1. And, we then represent the overviews of OPC techniques, which are used in automation and control systems, in Section 2.2. Finally, the problem statements are discussed in Section 2.3.

2.1. Model of Web Integration

A general architecture of Web integration consisting of three layers is shown in Fig. 1 [7]. The lower layer provides information from automation devices to controller level of an automation system. The upper layer is based on standard IT technologies such as Client-Server model, using Web Server as data source and Web browsers as clients. The middle layer contains the functionalities of business logic and performs as an application gateway between the upper clients and lower automation systems. The Web Server can be used to assign information from the automation and control systems to object models that can be accessed via COM/DCOM (Distributed Component Object Model). The most important problem is the clear data mapping between data and Web application. The data have different types and different semantic meaning. Using Web technologies for slow control systems, it means integrating a multitude of different technologies.

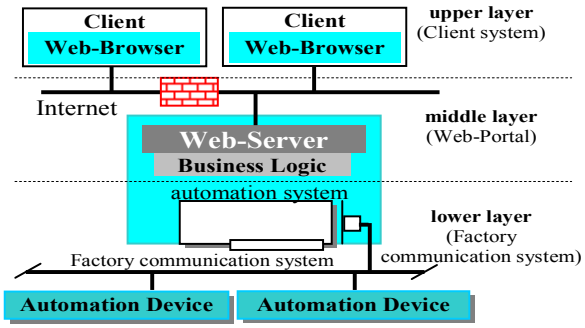


Fig. 1. Generic of Web architecture integration

2.2. Overview of Related OPC Techniques

OPC Foundation, formed in 1996, is independence, non-profit, industry trade association comprised of more than 300 leading automation suppliers worldwide. This section provides some overviews of related OPC technologies. Firstly, OPC DA [8] based on COM/-DCOM. This specification defines a set of standard COM objects, methods, and properties that specifically address interoperability requirements for factory automation, process control and condition monitoring applications, etc. The OPC DA leverages DCOM allowing the Client-Server applications to access plant floor via an Ethernet network distributed across the manufacturing enterprise. But, OPC DA applications are only compatible with cooperating applications-based Microsoft Platforms. Secondly, OPC XML-DA [3] defines a new specification to move the same type of plant floor data as existing OPC DA COM-based products. This provides vertical integration between the plant floor and condition, monitoring, maintenance, etc. using XML, HTTP, and SOAP industry standards. The OPC XML-DA provides better connectivity and interoperability for production management and enterprise applications such as MES, ERP, CMMS, EAM, and plant optimization that need to access to the plant floor data. In addition, it is complementary with products based on existing OPC DA Specification. It was specifically designed to allow existing OPC DA COM-based products to be wrapped by the OPC XML-DA interfaces and in effect support both interface from the same OPC Server. It is as a standard Web Service

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

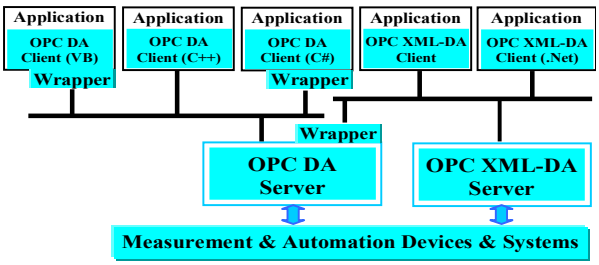


Fig. 2. The OPC DA and OPC XML-DA provide plant floor to manufacturing enterprise integration

interface for reading and writing data from and to plant floor automation systems [3]. The OPC DA and XML-DA provide plant floor to manufacturing enterprise integration as shown in Fig. 2. Thirdly, OPC Complex Data [4] initiative will provide a full way for the OPC Clients to read and decode any type of data from hardware I/O devices on the plant floor. Complex data mean that an OPC Item is defined as a structure. The item includes read-only information, run-time status, and writeable control points. Actually, the complex data contain complex data items that can include non-structured items, structure items, XML data, OPC binary, etc [4]. The OPC Complex Data specification defined two type systems that provide this level of capability, *XML Schema* and *OPC Binary*. Finally, OPC DX (Data eXchange) [12] has well-defined objects that are based on OPC COM-DA and XML-DA objects. The OPC DX addresses the simple mechanism for moving data between source and target (i.e. destination). It sets out the rules associated with the when, what and how of moving the data between end points. Moreover, it defines for dealing with exception conditions, the value to be written to the target or maintained at the target when good data are not available from the source. The OPC DX adds some key extensions by leveraging the OPC DA and OPC XML-DA standards to exchange data horizontally between peer level OPC applications. The OPC DX extents data access to enable server-to-server data exchange during run-time, independent of the real-time applications that support by Ethernet networks [13].

2.3. Problem Statements

As a big step for process control, the OPC Foundation created a new XML-DA to allow wrapping existent OPC DA that is widely successful standard based on COM/DCOM. Thus, PLC (Programmable Logic Control), DCS (Distributed Control System), HMI (Human Machine Interface) and other factory-floor software vendors use the OPC standards to compatibility and interoperability. OPC XML-DA solution provides more advantages that have been represented in Section 2.2. However, the OPC XML-DA solution has a big disadvantage that is to use XML data representation [3], [6]. XML data representation causes much more network traffic to transfer data. Moreover, data alignments are often required transport data in native representation. The OPC Complex Data based on using XML presentation that also requires big amount of memory and high intensity of memory management operations. In addition more CPU resources for transformation between native data representation and XML are required [9]. A further issue is the present unavailability of XML versions of OPC HDA (Historical Data Access) [10] and OPC AE (Alarms and Events) Specifications [11] that are typically required in scientific experiments. In [2], the authors provided the design and implementation of OPC XML-DA Server that allows OPC Clients to read and decode any type of data from measurements and control systems. However, they have not provided any the evaluation of performances. Additionally, the OPC XML-DA requires XML messages to be very descriptive about the data being transferred. For example, instead of "0.555" the record *<value xsi:type = "xsd:float">0.555</value>* is sent, that means the bandwidth is increased about 6 times or even more [6]. Besides these problems, the requirements to convert the data between XML representation and OPC binary representation make it to determine the required size of the memory buffer to hold the whole data. Additionally, the only fast possibility to transmit large amount of data is to use binary representation rather than XML representation. However, different platforms even different compilers of the same platform use different binary representation such as different floating formats, different string formats, etc. In addition, we need to aggregate a thousand of the existing OPC COM-DA products use in today with OPC XML-DA products into a distributed data acquisition and control system that can exchange horizontally data between COM-DA and XML-DA Servers. Another requirement is to allow OPC Clients to access to both the COM-DA and XML-DA Servers.

3. Framework for Design and Implementation

This section, we present a framework for design and implementation of *Universal DA Server*. This system allows aggregating the existing OPC COM-DA Servers and OPC XML-DA Servers to horizontally exchange data between them. The Universal DA Server is based on the OPC DX Specification.

3.1. The System Aspect Design

As we mentioned in Section 2, the OPC XML-DA was specially designed to allow existing OPC DA COM-based products to be wrapped by the OPC XML-DA interfaces. In addition, we expect a Gateway application for the OPC DA Servers, OPC XML-DA Servers, and other Gateways that Gateway provides a sound base for new applications and an easy migration path for the thousands of OPC DA products in use today. Therefore, we need a *Universal DA Server* based on OPC DX Specification that can allow integrating the DA Servers, XML-DA Servers, and Gateways to exchange data between them and also to allow accessing from the XML-DA Clients. The roles of the Universal DA Server are shown in Fig. 3. The Universal DA Server provides methods to allow XML-DA Clients to access to both the OPC DA and XML-DA Servers. Moreover, it is flexible to configure the access rights such as *write-only*, *read-only*, and *read-write* from the configuration client (i.e., remote client).

As shown in Fig. 3, multiple OPC DA and OPC XML-DA Servers can be configured as data sources. Data from source servers are handled according to the OPC DX standards. Using configuration client, we can configure the Universal DA Server to allow OPC XML-DA Clients to write data to the OPC Source Servers. The OPC complex data items of all configured server are mapped into the Universal DA Server address space. As in [2], the OPC XML-DA Server allows the OPC Clients to read and decode any type of data from hardware I/O devices. To implement the OPC DA Server supporting the complex data to allow the OPC Clients can read and decode any type from the hardware I/O devices, we design and implement OPC DA Server for supporting the complex data. Corresponding to our discussion, we also design and implement the Universal DA Server that can support the OPC Complex Data Specification. Hence, the Universal DA Server not only solves the problems to aggregate the OPC DA and OPC XML-DA Servers, but also permits the OPC Clients to read and decode any type of data from the hardware I/O devices. The DX Server is responsible for managing the data flow on a DX Connections from source to target. In general, it subscribes to data from the source and copies it to the target when it is received. The flow diagram of updating data to target items is shown in Fig. 4.

3.2. The Module Design

Universal DA Server can access to multiple OPC DA and OPC XML-DA Servers and it can be dynamically configured according to the OPC DX

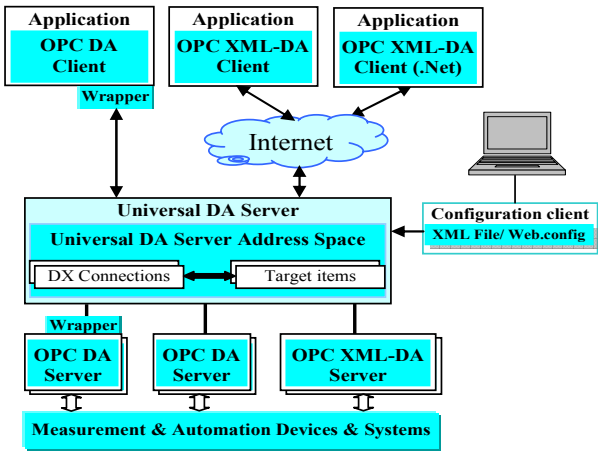


Fig. 3. A model of distributed data acquisition and control systems

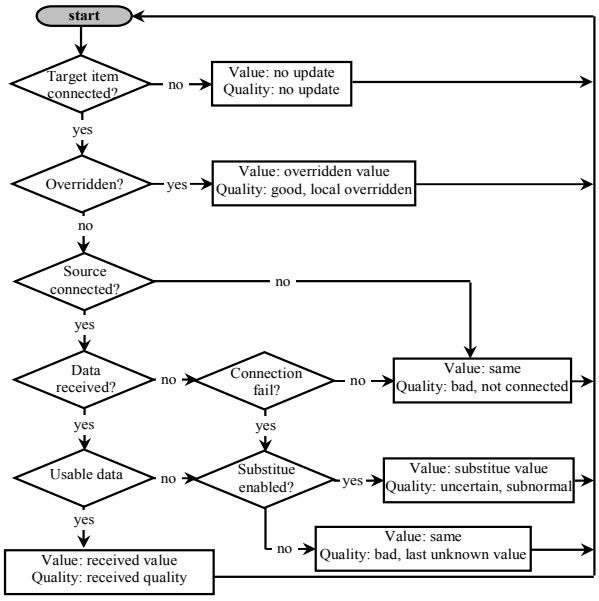


Fig. 4. The flow diagram of updating data to target items

Specification. The OPC Items of all configured OPC Servers are mapped into single address space that can be structured in any suitable way. Both read and write accesses are supported by the Universal DA Server. The design of the Universal DA Server to support multiple OPC Servers is shown in Fig. 5.

As discussed in Section 2.3, the *Universal DA Server* aggregates the multiple OPC DA and OPC XML-DA Servers as data sources that allow exchanging the data between them. The data are acquired from the source server depends on the type of the source servers such as OPC DA, OPC XML-DA. In general, the mechanism is referred to as a Subscription. Subscribing to the data is supported by *Callback* in COM-DA and by the *Subscribe* service in XML-DA. In

COM-DA, the Universal DA Server creates one or more groups for each source server. With defined groups, the Universal DA Server adds data items to them to indicate the source server which items to access. In XML-DA, the Universal DA Server uses *Subscribe* and *SubscriptionPolledRefresh* services to acquire source data. The Subscribe service allows the clients to define a single operation, a set of items for source server to access. As shown in Fig. 5, the Universal DA Server consists of three components that allow exchanging the data. The *COMModule* exposes OPC DA-interface and DX-configuration interface implemented as a COM-interface. The *SOAPModule* handles SOAP requests and exposes both the OPC XML-DA interface and the DX configuration interface by using WSDL (Web Services Description Language) [14]. The *DXModule* contains major parts of the DX Server functionality and it marshals data to and from the OPC Servers. The *DXModule* can create new connections, remove connections, and modify the status of each connection. The Universal DA Server is required that DX-Connections can be modified and controlled by using SOAP and COM Clients

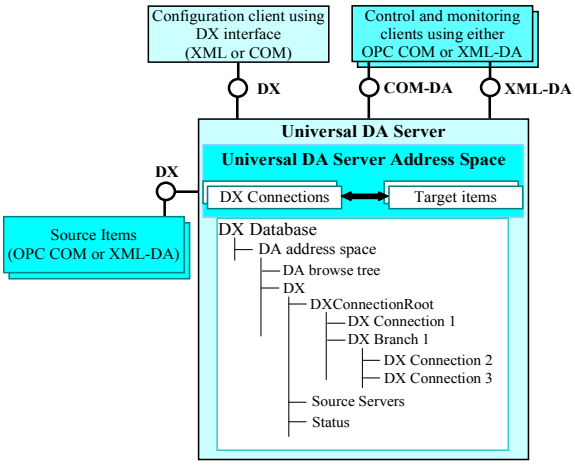


Fig. 5. The design of Universal DA Server to support multiple OPC DA and OPC XML-DA Servers

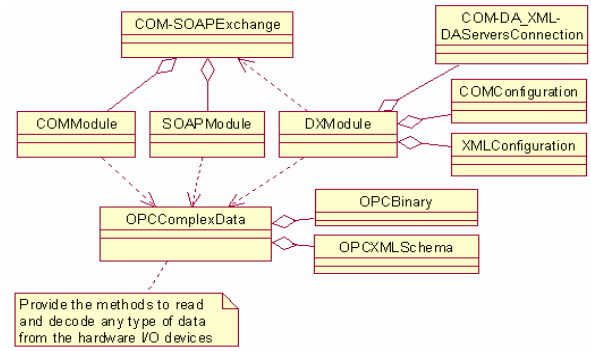


Fig. 6. The module design of Universal DA Server

simultaneously. Consequently, it requires solving the inconsistencies and synchronization problems.

The design of the modules for the Universal DA Server to aggregate multiple OPC COM-DA and XML-DA Servers into the system that can exchange data between COM-DA Servers and XML-DA Servers is shown in Fig. 6. To support OPC Clients to read and decode any type of complex data from hardware I/O devices, *OPCComplexData* class is used in the Universal DA Server. This class provides the methods to represent and convert the complex data that complex data types are defined as dictionaries. Thus data types are described and presented by using XML schema. Besides the combination of multiple COM-DA and XML-DA Servers, the Universal DA Server based on OPC DX Specification that provides a means to exchange the data between the COM-DA and XML-DA Servers. As shown in Fig. 6, the *COMSOAP-Exchange* class aggregates the *COMModule* and *SOAPModule*, and it is used by the *DXModule* class. This class handles the SOAP requests from OPC XML-DA Clients by using WSDL. It exchanges data between the OPC COM-DA and XML-DA Servers. Thus for all active DX Connections, the *DXModule* keeps up volatile run-time objects as well. To marshal data the *DXModule* is responsible for updating necessary status items that are associated with each DX Connection. These status items can be used to observe the current state of the connection and data flow. The status connection information changes constantly during the runtime data exchange. To continuously describe these modules, we represent the class design of these modules, *COMModule* and *SOAPModule* as shown in Fig. 7. These modules provide a means and methods for processing the COM objects or SOAP objects depending on the OPC Clients (i.e., either COM-DA Clients or XML-DA Clients). The *SOAPModule* is used not only for the XML-DA, but also for existing COM-DA by using wrapper. Therefore, this module is composed of the *NET-XMLWrapper* and COM-*DAWrapper* class. The *OPCDACache* that provides a means to solve the large control network problems is aggregated in the *COMModule* class.

As aforementioned, the Universal DA Server that supports not only to exchange data between OPC Servers, but also to allow the XML-DA Clients to access to both the COM-DA and XML-DA Servers is based on the OPC DX Specification. Especially our system is designed as a Web Service, which allows the clients running on Microsoft Platforms and non-Microsoft Platforms to access by using the XML leverages. Moreover, the data are converted and represented as binary data representation that provides

a means to effectively utilize the memory, resources, and bandwidth of the systems.

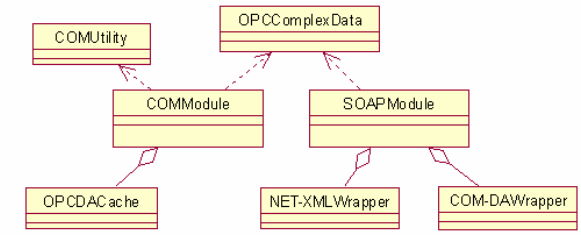


Fig. 7. The class diagram of the COM and SOAP modules

3.3. XML based Description Model

Application-based framework for industrial system relies on an XML based set of interface descriptions. These descriptions represent interaction schemas between specific applications and specific contexts. Conversations between the schemas are according to transformation rules. The application framework used content model to request data source and data formats for the values that have to be read out of the factory communication systems and their components. Since the XML schemas contain information on the data types, value ranges, display information, access paths, parameters, etc. The application can determine and represent values and related information on the client-side. Therefore, using the WSDL files we can construct a standard XML-based application. The XML-based content model [15] consists of a distributed set of XML files, schemas, and transformation rules. In order to prevent the multi-definitions, the files are linked together. As discussed, the Universal DA Server can be configured by configuration clients with use of XML schema or WSDL files. This feature guarantees the flexibility and effect of our system.

3.4. Evaluations of Universal DA Server

According to our design and implementation, the Universal DA Server provides a full way to combine more than thousand of the existing OPC DA with the OPC XML-DA Servers and to exchange the data between them. It allows the OPC XML-DA Clients to read and write the structured data to both the OPC DA and OPC XML-DA Servers. The important ability is to be able to make such server deployments that allow porting Web Service part for non-Window Platforms also, particularly to Linux. The Universal DA Server also ensures the OPC COM-based DA components that have already been developed and tested to be reused as effectively as possible. Besides, the Universal DA Server provides an opportunity to configure and monitor connections simultaneously using Web Service interface and COM interface. In fact, this feature is used for Window Platforms. In brief, the design of Universal DA Server guarantees the portability, component reuse, system extension abilities, and performance of data exchange.

4. Database Connection and Representation

The evaluations of binary data representation are exposed in Section 4.1. *XML Libraries* comparisons are then discussed in Section 4.2. In Section 4.3, we represent the database connection between Universal DA Server and database for open issues.

4.1. Binary Data Representation

As we mentioned in Section 2.3, the only fast possibility to transmit the large amounts of data is to use binary data representation rather than XML data representation. In addition, different platforms and even different compilers of the same platform use different binary representations such as different bytes order for representing multi-byte data, different floating point formats, etc. Thus, universal standards should be chosen to transport data between different platforms. Therefore, we must investigate and compare the current available binary encoding standards. The fundamental approach to solve the bandwidth problem is using binary data representation, which is integrated into XML such as BXML [16], BXSA [17], etc. Several proposals are available to satisfy these conditions such as SOAP message [18] with attachment or the HTTP message using XLink (XML Linking Language) [19]. The Universal DA Server will respond the XML-DA Clients with SOAP message with all the data replaced by XLink references. To incorporate binary data into SOAP message, the WS-Attachment [20] technology is used. In addition, XLink is used to link different parts of compound message together. Addressing of multi-groups can be done in XLink.

4.2. XML Libraries Comparison

When design and implementation of an application system, we have to solve existing problems of weak ground. If there is new software available made by different manufacturers, the best thing to do are benchmarks. To achieve a better basic for a decision to select the best XML library, several benchmarks ware performed to evaluate present XML libraries. There are many available XML library benchmarking projects such as XMark [21], XML Benchmark [22], SAX Parser Benchmark [23], etc. However, they are not very well applicable for OPC XML-DA. In addition,

they tested only one or two aspects of XML processing, with some predefined sequences of XML data. To provide fast and reliable OPC XML-DA or Universal DA Server solution, a fast multiplatform XML toolkit is required to process different types of XML files at high speed. Also, we have to investigate some supports including XML-binary Optimized Packing [24], XML Encryption [25], XML Signature [26], [27], XML Transformation [28], and some other. The benchmark results of five different libraries are shown in Fig. 8 [29]. On some platforms, multiple compilers are available and should be compared, since the performances of the XML libraries are very important for the overall system performances. The performance tests can be divided into the following phases of XML processing such as schema validation, XSL Transformation, XML Security, etc. As shown in Fig. 8, the *LibXML* is effectively high performance and it is selected to apply to our system.

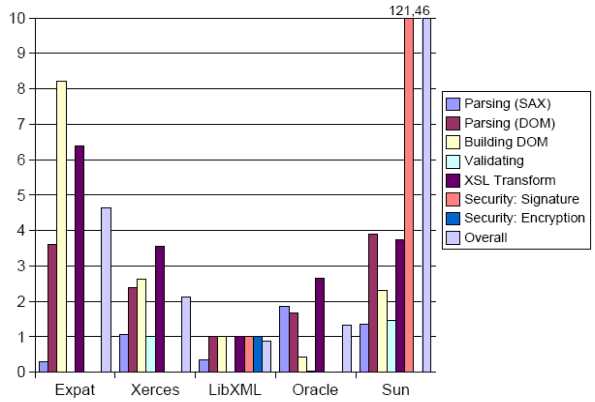


Fig. 8. Benchmark results of five different library combinations

4.3. Connection between Universal DA Server and Database

To provide an open and flexible system for implementing and aggregating the OPC HDA [10] into our system, the connection between Universal DA Server and Database is investigated and provided. Standard SQL (Structure Query Language) as basic query language for database searches in many cases is not efficient. Consequently, each database has its own extension to SQL such as PL/SQL for Oracle, PL/pgSQL for postage-SQL. However, with each SQL solution is not independent platform. In fact, we expect to achieve the solution with independent platform by using XQuery. But in large applications the XQuery is still too slow and too memory consuming as shown in Fig. 9 [6]. When Universal DA Server is defined as a Web Server and supports to store and exchange the data from Web Server to the OPC Clients, it should be designed for expanding the connection between the Universal DA Server and Database. To build a full system-based OPC techniques, we will investigate the ability to extend and integrate our system when adding new OPC Specifications or combining the existent OPC Specifications. However, the situation of XQuery will change rapidly in near future due to efforts put into the further development of XML.

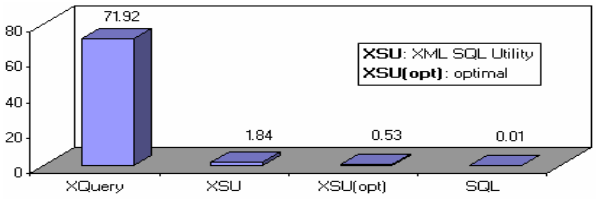


Fig. 9. The comparison between query languages

5. Security Aspects

The security is very important position in application when using both Ethernet and Internet. In order to achieve required security criteria, the concepts and solutions developed for General IT system have to be applied. Appropriate technologies like encryption, SSL (Secure Socket Layer) technology, HTTP-S (Secure HTTP), certificates and digital signature should be used. The solution for our system has to guarantee the security to both COM-DA and XML-DA. In addition, the described solutions of bandwidth problems raise new problems that are security problems. In the standard cases, the HTTP-S can be used to protect data. However, for multicasting data connections HTTP-S is not available and some other mechanisms must be used. A proposal is to use an authentication server that will use SSL private/public keys for authorization and generate symmetric session keys. Moreover, it also proposed to use the internal XML security approach for the control connections instead of the HTTP-S protocol, being described in XML Encryption [25], XML Signature specifications [26], [27], and XML Decryption [30]. The XML distributed signature approach is represented in [31]. This solution is really optimal to reduce the data transfer between servers and clients, and ensures thin clients. With COM-DA, the security based on DCOM Security that distinguished between connection security, per call security, and per packet security is exposed in [32], [33]. Another aspect, the security concepts with two sections, one for Intra/Internet and one for the field level are proposed in [34]. As we already discussed, the security in our system has to ensure to guarantee for both Ethernet and Internet. Moreover, there are no currently authentication protocol designs for the factory IT environments that require security as well as effectively.

6. Conclusions

In this paper, we have already introduced the design and implementation of modern distributed acquisition data and slow control systems based on OPC techniques. Our system allows the existing OPC COM-based DA and OPC XML-DA to aggregate into a system that permits to exchange complex data between OPC Servers (e.g., DA Servers, XML-DA Servers) also, the OPC Servers and Clients. This is a solution to provide a sound base for new applications and an easy migration path for the thousands of OPC DA products in use today. To overcome the performance limitations of XML representation, we have proposed to use the binary data representation in the Universal DA Server. Besides, we have investigated and compared the XML Libraries to choose XML technology as a candidate in our system. And the *LibXML* should be used to resolve the system requirements (e.g., any platform, acceptable performance, etc.). Moreover, our system can run on Window Platforms and non-Window Platforms, particularly to Linux. To configure the Universal DA Server, we can use remote clients that really ensure flexibility and optimization of the system. In addition, the security aspects are discussed in our works. These aspects provide seamless security solutions in big systems that are communicated by using the Ethernet and Internet environments. They also provide more security information for technical-level readers.

7. Acknowledgment

The authors would like to thank the NARC (Network based Automation Research Center) Center of University of Ulsan for its technical and financial support for our work described in this paper.

References

[1] D. W. Holley, "Understanding and Using OPC for maintenance and reliability applications", *Journal*, *IEE Computing and control engineering*, Mar. 2004.

[2] V. V. Tan, D. S. Yoo, and M. J. Yi, "Design and Implementation of Web Service by Using OPC XML-DA and OPC Complex Data for Automation and Control Systems," *Proc. of the 6th IEEE International Conference*, p. 263, Sept. 2006

[3] OPC XML Data Access Specification version 1.01, [Online]. Available: http://opcfoundation.org/, Dec. 2004

[4] OPC Complex Data Specification version 1.0, [Online]. Available: http://opcfoundation.org/, Dec. 2003

[5] S. Chilingargyan and W. Eppler, "High speed data exchange protocol for Modern distributed data acquisition systems based on OPC XML-DA," *Proc. of the Real-time Conference, 14th IEEE-NPSS*, pp. 352-356, Jun. 2005

[6] W. Eppler, et al, "New Control System Aspects for Physical Experiments," *IEEE Transaction on Nuclear science*, vol. 51, no. 3, pp. 482-488, Jun. 2004.

[7] M. Wollchlaeger, P. Nweumann, and Th. Bangemann, "Web service for remote maintenance of fieldbus base automation systems," *Proc. of the IEEE International Conference*, pp. 247-252, Oct. 2002

[8] OPC Data Access Specification, [Online]. Available: http://opcfoundation.org/, Dec. 2004

[9] F. Bustamante, G. Eisenhauer, K. Schwan, and P. Widener, "Efficient wire format for high performance computing," *Proc. of the ACM/IEEE Conference Supercomputing*, p. 64, 2000

[10] The OPC Historical Data Access Specification, [Online]. Available: http://www.opcfoundation.org/, Dec. 2003

[11] The OPC Alarms and Events Access Specification [Online]. Available: http://www.opcfoundation.org/, Oct. 2002

[12] The OPC Data eXchange Specification version 1.0. [Online]. Available: http://www.opcfoundation.org/, March. 2003

[13] H. Xiaohong and H. Sunhong, "OPC DX and Industrial Ethernet glues fieldbus together," *Proc. of the Control, Automation, Robotics, and Vision Conference*, vol. 1, pp. 562-567, Dec. 2004

[14] Web Services Description Language (WSDL) version 2.0, March, 2006 [Online]. Available: http://www.w3.org/TR/wsdl20/

[15] M. Wollchlaeger and T. Bangemann, "XML based Description Model as a Platform for Web-based Maintenance," *Proc. of the IEEE Conference*, pp. 125-130, Jun. 2004

[16] C.S. Bruce, "Cubewerx Position Paper for Binary XML Encoding", [Online]. Available: http://www.cubewerx.com/main/HTML/-Binary_XML_Encoding.html

[17] K. Chiu et al, "A Binary XML for Scientific Application", [Online]. Available: http://www.cs.indiana.edu/~chiuk/bxsa.pdf

[18] SOAP Version 1.2 Recommendation 2003/6/24. [Online]. Available: http://www.w3.org/TR/SOAP

[19] XLink (XML Linking Language) version 1.0, 2001/6/27. [Online]. Available: http://www.w3.org/TR/xlink/

[20] SOAP message with Attachments, Dec. 2000, [Online]. Available: http://www.w3.org/TR/SOAP-attachments

[21] XMark, an XML benchmark project, [Online]. Available: http://monetdb.cwi.nl/xml/index.html

[22] XML Benchmark [Online]. Available: http://www.sosnoski.com/opensrc/xmlbench/results.htm;

[23] SAX Parsers Benchmark [Online]. Available: http://piccolo.sourceforce.net/bench/html

[24] XML-binary Optimized Packing, W3C Recommendation, 2005/01/25, [Online]. Available: http://www.w3.org/TR/2005/REC-xop10-20050125/

[25] XML Encryption Syntax and Processing: W3C Recommendation 2002/10/10, [Online]. Available: http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/

[26] XML-Signature XPath Filter 2.0:W3C Recommendation, 2002/11/08. [Online]. Available:http://www.w3.org/TR/xmlenc-core

[27] XML Signature and Processing: Recommendation 2002/2/12. [Online]. Available: http://www.w3.org/TR/2002/REC-xmlenc-core-20021210/

[28] XSL Transformation (XSLT) version 1.1, 2001/8/24. [Online]. Available: http://www.w3.org/TR/2001/WD-xslt11-20010824

[29] S. Chilingarian, "Fast, Multiplatform XML Toolkits Comparison. Functionality, Speed and Memory Usage," [Online]. Available: http://prdownloads.sourceforge.net/xmlbench/features.pdf, http://prdownloads.sourceforge.net/xmlbench/benchmark.pdf

[30] Decryption Transformation for XML Signature: W3C Recommendation, 2002/10/10. [Online].Available: http:www.w3.org/TR/xmldenc-decrypt/

[31] Miyauchi, "XML Signature/Encryption – The basic of Web service security," *NEC Journal of advance technology*, vol. 2, no. 1, 2005

[32] OPC Security Custom Interface Specification, [Online]. Available: http://opcfoundation.org/, Oct. 2000

[33] D. Dzung, M. Naedele, T. P. V. Hoff, and M. Crevatin, "Security for Industrial Communication Systems," *Proc. of the IEEE International Conference*, vol. 93, no. 6, pp. 1152-1177, Jun. 2005

[34] A. Treytl. T. Sauter, and C. Schwaiger, "Security Measures in Automation Systems – a practice-oriented approach," *Proc. of the 10th IEEE Conference*, vol. 2, pp. 847-855, Sept. 2005

Node-Oriented Modeling and Simulation of IP Networks

Nanjun Li
Hasso Plattner Institute at University of Potsdam
Potsdam, 14482, Germany
Nanjun.Li@hpi.uni-potsdam.de

Abstract

IP networks consist of autonomous nodes that perform routers' and hosts' functionalities. Based on this fact, this paper proposes a node-oriented IP network model and a new packet-level and stochastic discrete event-driven simulator – Visualized IP Network Simulator. The XML is used to encode node and network topology for scenario. Each individual node is completely autonomous and has its own built-in routing table, queue and service routines. They are distributed in different domains.

With respect to the real network addressing and subnetting schemes, VINS enables large-scaled network simulation with complex topology, big node population and crossed IP packet flows. New evaluation methodology emerges in its runtime system report, calculating the performance on node, flow and network levels, providing a better insight into the IP networks.

1. Introduction

An appropriate model helps people study the real systems, build up simulations, find problems and seek solutions. Modeling IP networks is to abstract the key properties from the real ones. Among existing IP network models and simulators, there are a number of drawbacks. For example, in NS-2 [1], the queues are wrongly put on the links, which should have been setup within the nodes. The simulation based on this model prevents people from calculating the nodes' performance. In addition, the network addressing schemes is not proper, so the multi-domain simulation is difficult. These drawbacks have been recognized (e.g., [2]), and a better simulator might be expected.

This paper starts with a brief review over some well-known simulators, their approaches and problems, and proposes a new model for IP network simulation. With this model a new simulation tool – Visualized IP Network Simulator (VINS) is implemented. We will introduce its design, the encoding methods in building up scenarios and the evaluation methodology. An example is made to demonstrate VINS' strength, features and the new insight it can provide.

2. Background and Aims

Simulation of full-functional TCP/IP network is very difficult. There is a number of existing network simulators using different abstractions / approaches with advantages and disadvantages. NS-2 [1] and REAL [3] are designed in the early stage using a traditional approach constructing simulation scenarios with both nodes and links ("lines" in REAL). The link is improperly associated with a queue, queue-limits, bandwidth etc., which should have been associated with each individual node.

OPNET Modeler [4] is a commercial product, using a series of related editors to build scenarios. Users can assign the nodes and applications with very specific attributes than with NS-2. Its problems might be: the building up of scenario is time consuming; the internal simulative process is not visible.

OMNeT++ [5] is a free, relative new network simulator. Its scenarios are built with modules that are integrated with a FreeBSD kernel, thus avoid the implementation of TCP/IP simulation core; the inter-domain simulation issue was partially solved [6]. In comparison with the earlier ones, its simulative result might be more valid. However, the tight integration with a single FreeBSD kernel made it not flexible to run heterogeneous protocols; calculation on individual node can hardly be made; the scenario topology of is still limited.

As a new network simulator, VINS is based on the classical queuing theory (e.g. [7]), designed and developed with a set of distinctive aims: its model shall be close to the real networks; its simulation core follows the 4.4BSD kernel [8]; the simulative results must be consistent and traceable on every node and application; it must support large-scale network simulation and be able to run heterogeneous protocols; it shall be extensible for continuous

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

development; it shall be "visualized", which could bring a consistent big picture for the users who are doing network modeling, teaching and research.

3. Modeling IP Networks

Not only an IP network is an autonomous system (AS), but also is each individual node. An individual node has its own built-in routing table, queues and service routines. It routing table defines its connectivity relationship with other nodes, which is a part of network topology. Queues are used to store packets (can be viewed as service request) that cannot be served immediately; in 4.4BSD, it refers to the "Interface Queue", with a capacity `ifq_maxlen` default to 50 [8].The service routines, includes a forwarding engine and packets time-to-live (TTL) control. TTL will be decremented on the node, and over-aged packets will be discarded. We may take a survey on a software router based on BSD [8] system. A behavioral diagram is made with Petri-net [9], shown in Figure 1:

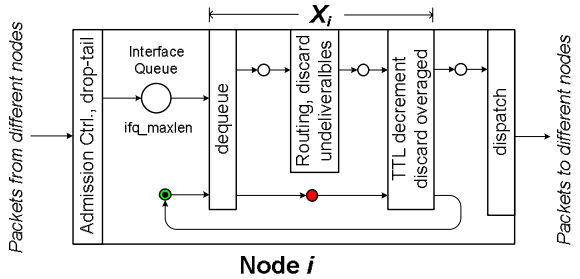


Figure 1 Behavior of a Node

The X_i denotes the "mean service time" within the node. A packet's service time is counted as the duration since it is dequeued to its departure. The actual service time for a packet is stochastic, which depends on the CPU's random access and the routing table size, algorithms and so forth. We assume this process takes a certain probability density function (PDF).

The admission control will drop the incoming packets if the queue is full and the lost counter on this node shall be incremented (drop-tail). There are also other queue management technologies such as Random Early Detection (RED) [10].

An IP packet records its source and destination addresses, while a node's routing table only enlists its neighbors that may not contain the destination. So the node can only forward the packet to the most hopeful one that might move the packet to the destination. This job is done by the forwarding engine and requires global agreements on addressing and subnetting schemes. Today's Internet address is 32-bit long (IP v4), prefixed with a network identifier (for both Classful [11] and CIDR networks [12], Figure 2). In a Classful network, a class C router with IP address 12.34.56.78 has a 24-bit long network identifier (12.34.56.xx) with mask 255.255.255.0.

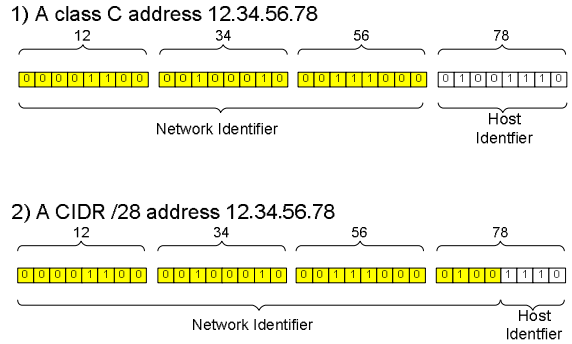


Figure 2 Internet Addressing Schemes

Classful networking is not flexible enough in the real world: e.g., to build a local area network (LAN) consisting of 15 hosts, a class C address is required as the gateway, which can actually handle 255 hosts. So class C addresses had been nearly used out in the early 1990's and CIDR networking was invented, in which the 8-bit class was dissolved and replaced with bit-width subnetting scheme. In CIDR's solution, a LAN with 15 hosts only needs to apply for a "/28" CIDR block, with mask 255.255.255.240.

Classful subnetting / addressing scheme is adopted in VINS since it is enough for simulation. Figure 2 presents the forwarding algorithm:

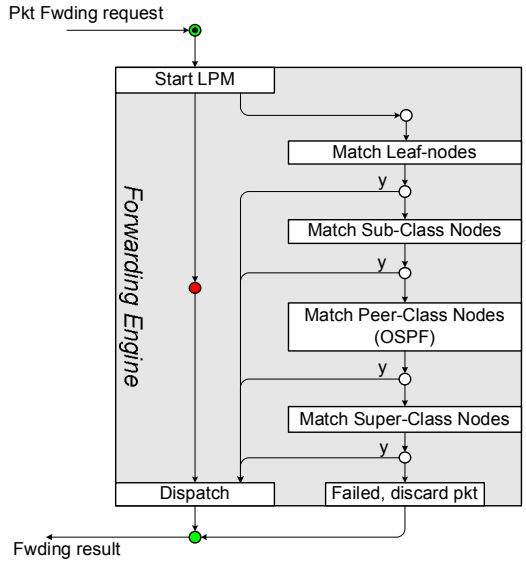


Figure 3 VINS Forwarding Engine

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Routers perform Longest Prefixes Match (LPM, [13]) to find the best matching node from its routing tables; if there is no exact or subnet border node to match the packet's network prefix, it may take the Open Shortest Path First (OSPF, [13]) algorithm to find a closest peer-class neighbor to forward; and if it has no peer-class neighbor, it may pass it to a super-class node who is not in its super-domain (to avoid packet bounce-back), or simply discard it.

With the models and discussion above, a scenario of IP networks can be built as shown in Figure 4, which consists of 10 nodes distributing in 4 domains (including a super one: 12.34.xx.xx):

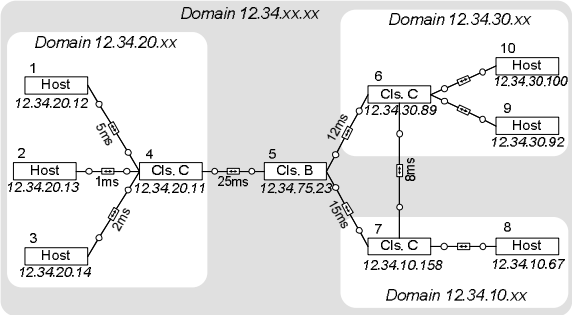


Figure 4 A "Large-Scaled" IP Network Scenario

This model can be termed as **Node-Oriented**, with respect that the service on IP Layer takes place within the nodes (routers and hosts). Meantime, the physical transmission service is provided by "**transport server**", which is a unit between each two nodes, denoted as an active component assigned with a **transport service time**.

4. Encoding Nodes for Scenario

To simulate IP networks we have to encode the autonomous nodes in meaningful structures that both human and simulator can understand. The eXtensible Markup Language (XML, [14]) is selected as the encoding language. A number of node's properties are selected for the simulation, as defined, abbreviated and denoted in the following:

Name is used to identify a node, which should be a string in standard ASCII codes. The name of a node shall not be duplicated in the same scenario, because in VINS a node's routing table records the neighboring nodes by names.

Address specifies a node IP address in dotted-quad format, e.g., 12.34.56.78. As well, the IP address shall not be duplicated in the same scenario.

Type specifies the type of a node. VINS currently supports 5 types of node with enumerable values: {CONSUMER, SUPPLIER, HOST, CLASS_C CLASS_B, CLASS_A}

A HOST performs full functionalities as a leaf-node in the IP networks, where applications can mount on and bind to a certain port. This makes VINS extensible for future development including the transport layer simulation (TCP and UDP). For convenience, 2 special types of HOST node are designed to generate IP packet flows: 1) CONSUMER simply receives and discards arrival packets; 2) SUPPLIER simply produces packets destined to a CONSUMER.

A node with CLASS simulates the router in the IP networks. It queues incoming IP packets from unknown nodes and provides services, including routing, TTL decrementing and relaying them to its neighbors, in hope to move these packets to their destinations (leaf-node).

Table 1 maps VINS node types to real network address classes and CIDR prefixes:

Table 1 VINS Nodes Types:

VINS Node Type	Network Bits	Host Bits	Address Class	CIDR Block Prefix
CLASS_A	8	24	A	/1 - /31
CLASS_B	16	16	B	
CLASS_C	24	8	C	
HOST SUPPLIER CONSUMER	32 bits	0	Host	/32

Pos records the screen position of a node, specified with two decimal numbers. User can change the layout of nodes as they prefer.

Cap. defines the capacity of the queue of a node, limiting the maximum numbers of packets in the queue. When the queue is full, new arriving packets will be dropped.

X is the mean service time, an attribute of a node's service routine. In a real router it is relevant to the length of its routing table, algorithms, CPU speed and

etc. It is the average time that a node takes (unit: millisecond) in a packet service.

Dev is the deviation factor, an attribute of a node's service routine. This factor is used in the transformation from normalized distributed random numbers to non-normalized ones for each individual node, to differentiate their characteristics in service.

PDF is the abbreviation of Probability Density Function, an attribute of the service routine identifying the service time distribution. Currently two options are supported: Uniform and Gaussian distributions: {GAUSSIAN, UNIFORM}. X, Dev and PDF are used together in the transformation calculation.

RtTab encodes the routing table (in BSD's context, [8]) of a node by recording the names of its neighboring nodes. A node can only relay packets to one of the node enlisted in its RtTab. Currently VINS does not support multicasting or broadcasting.

Since the transport service time in the Physical Layer between two nodes is always considerable, users are recommended to use expressions to denote this time, for example:

<RtTab>A|10,B|15,C|20</RtTab>

It means the transport service time to node A takes 10 ms, to B takes 15 ms and to C takes 20 ms. Thus they will be counted in simulation.

Further, the link between two nodes can be either duplex if two nodes mutually enlist each other in their routing tables; or simplex, if one enlists the other's name while the other does not.

Destination is the name of a CONSUMER. It is assigned as an attribute of a SUPPLIER, which generates packets and stamps their destination.

Figure 5 presents a segment of scenario code in XML as an example:

```
...
<Node>
<Name>D</Name><Address>12.34.77.1</Addres
s><Type>CLASS_C</Type><Pos>830,741</Pos><
Capacity>30</Capacity><Mean>44</Mean><Dev
>51.200</Dev><PDF>GAUSSIAN</PDF>
<RtTab>G|1,E|3,F|2,N|10</RtTab>
</Node>
...
```

Figure 5 Code Segment of VINS Scenario

The IP packets are simulated with structured entities – IPSgmt. It is very close to the real ones that defined in BSD, shown in Figure 6:

```
struct IPSgmt
{
unsigned long src, dest; // 32bit addrs
long ttl;    // time-to-live
struct ProtoHeader
    {
    int ProtoId; // TCP, UDP etc.
    unsigned int srcport, destport;
     ...
    } prthdr;
};
```

Figure 6 Data Structure for IPSgmt

When a SUPPLIER creates a new packet, it fills the source and destination address fields `src` and `dest`, stamps the initial TTL and protocol header (for VINS' applications).

5. VINS Design and Implementation

VINS is a WIN32 application. Its software is designed with UML design patterns [15] and implemented with C++. The simulation core and other utilities are separated in source code, which made VINS easy to be ported onto other platforms.

There are four major components comprising VINS:

- XML-interpreter
- Scenario-manager
- Graphical User Interface (GUI)
- Random Value Generator (RVG)

The system architecture can be presented with a block diagram [9], as in Figure 7.

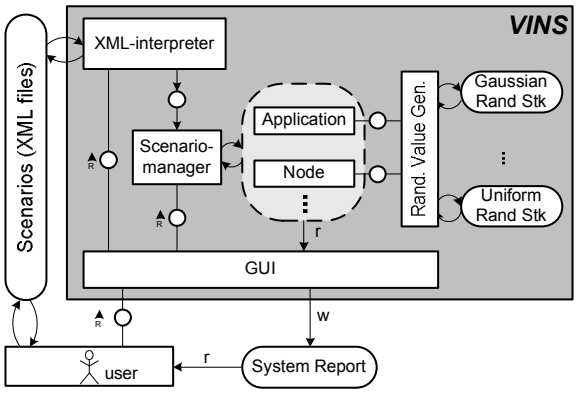


Figure 7 VINS Design

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

The **XML-interpreter** reads the scenario files (*.xml) from the hard disk, decodes the data and passes them to the Node-manager.

The **Scenario-manager** verifies if the scenario is ill-constructed and generates the software objects for simulation if no error is found. The simulation cannot be performed if one of the following cases in a scenario file is detected:

- Duplicated names;
- Duplicated IP addresses;
- Node references (in RtTab or Destination) that do not exist.

Scenario-manager keeps silent if no error is detected and enables the "Start" button for simulation. Otherwise, it will pop up a warning box listing all errors found.

The **GUI** of VINS is implemented with Microsoft Foundation Classes (MFC, [16]). Besides a user control panel, it provides an animation area where to visualize the simulation including the network topology, nodes' runtime activities, states of queues and some statistical data.

The service time for each individual packet at time t is stochastic and denoted as $X_i(t)$, which is the time this node shall sleep for simulation.

However, the generation of pseudo-random values by the computers is a complex process which takes unpredictable time and may influences the accuracy of timing and statistics, e.g., the Box-Muller Transformation [17]. A solution is to split off a process to pre-generate the normalized random numbers and push them into different stacks as PDF offers: {GAUSSIAN, UNIFORM}. When a node requires a random number, it simply pops one up from a stack. This mechanism is implemented with **Random Value Generator** (RVG). Thus the calculation of $X_i(t)$ needs only a linear transformation:

$$X_i(t) = X_i + Dev_i \cdot rand(t, PDF_i) \qquad (1)$$

X_i.is the mean service time on node i; Dev_i is its deviation factor; PDF_i. is its PDF option, all of which are specified in the scenario file. $rand(t, PDF_i)$ is a normalized (Dev:=1.0) random popped from one of the RVG stacks. If $X_i(t)$ is negative, the node will discard it and take 0 instead (no sleep).

Currently the timing accuracy in VINS is 1 ms, promoted from the regular 15ms (if use the regular *Sleep* function). This achievement is made by linking VINS with Microsoft Multimedia Timer Library (Winmm.lib and Mmsystem.h).

6. Simulation and Evaluation

VINS runtime environment requires Microsoft Windows-98 or higher. A package is available at [18], including the executable file and some scenarios.

In this section an example is made to demonstrate the strength of VINS and the insight it provides.

6.1 Build and Load a Scenario

Currently user can build a scenario with any text editing tool following VINS encoding rules. In future a graphical scenario editor may be developed in parallel.

Let's take "test.xml" in the package as an example. In this scenario there are 25 nodes: 7 SUPPLIERs, 7 CONSUMERs and 11 routers with different classes, including a deliberately isolated one (node Z). Thus there are 7 IP flows including an unreachable one going across the network (VINS can highlight their routes with moving bullets):

```
A->C->H->J
B->C->G->D->F
K->L->M->G->D->F
U->L->M->R->Q->S->T
V->S->Q->R->G->D->E
W->N->D->E
X->Y->M->R->Q->S (unreachable)
```

In the animation area users can have the scenario's overview: network topology, nodes' activities, states and three primary counters (put, loss and discard). As the meantime, VINS maintains a big number of counters so that a detailed system report can be generated at the simulation's runtime. This report is the statistical data calculated according to the counters. Calculations are on three levels: node, flow and network. The results are categorized in three tables (a sample is shown the Appendix).

Figure 8 is a screenshot of the simulation. The presence of some physical transport servers and as well the transport service time has been switched on (on the links from node R and Q):

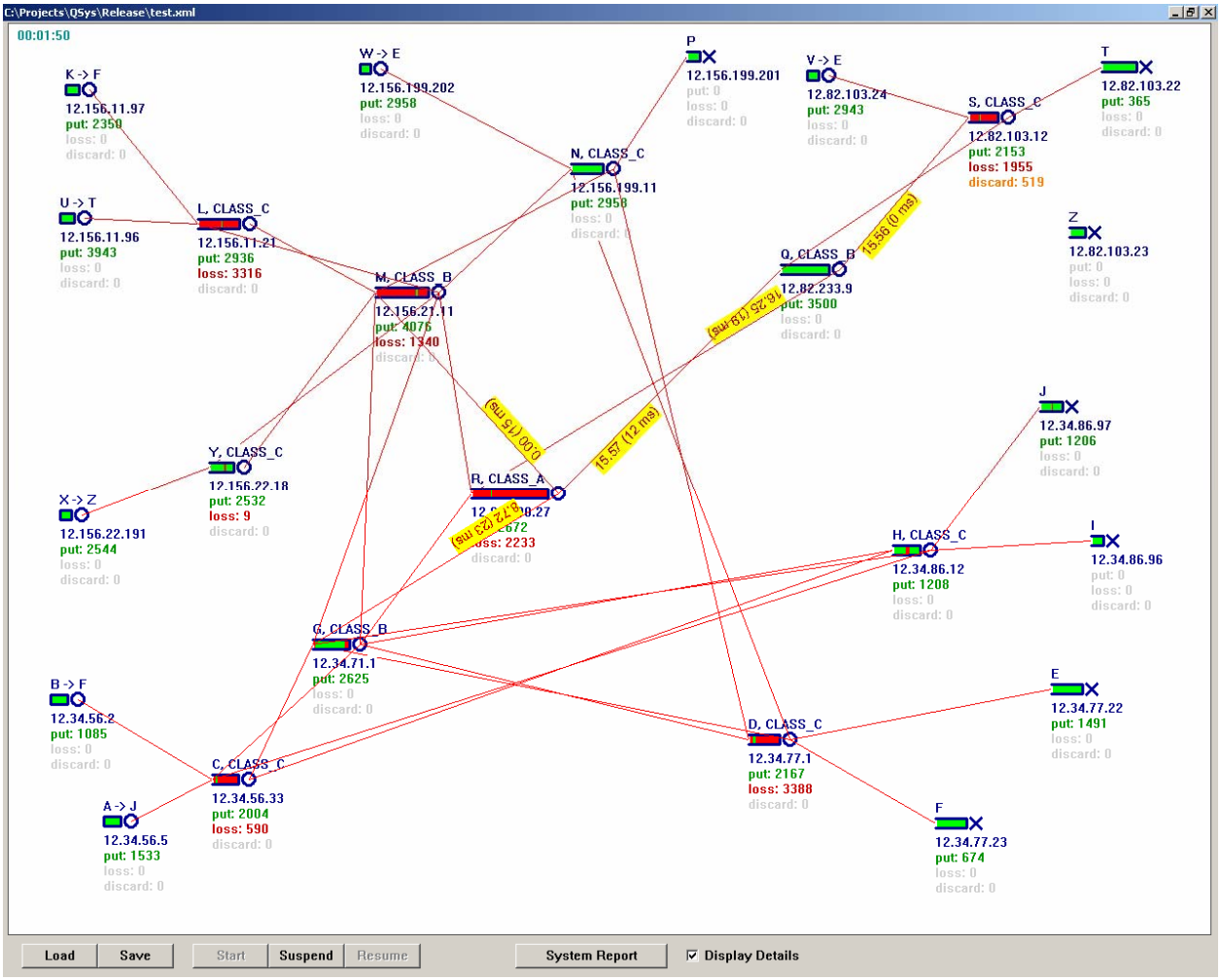


Figure 8 VINS Simulation Screenshot

6.2 Node Statistics

As mentioned, there are three primary counters displayed in the animation area: put, loss and discard. They are defined in the following:

Put is the total number of packets that have been served by this node (generated, relayed or consumed) since the simulation started.

Loss is the total number of packets that have been dropped by the node due to queue's overflow since simulation started.

Discard is the total number of packets that have been discarded by the node due to destination unreachable or TTL expired, since the simulation started.

Besides, VINS calculates the some other variables for node performance evaluation, abbreviated, defined and denoted in the following:

Arr.R. is the Arrival Rate, the average number of arrival packets at a node in one second, including those enqueued and dropped due to queue overflow, denoted as $\mathbf{A_i}$

Serv.R. is the Service Rate, a measure for the best performance a node can achieve. It equals to the reciprocal of its mean service time with unit [1/sec], denoted as $\mathbf{B_i = 1/X_i}$, where $\mathbf{X_i}$ is the mean service time.

Dept.R. is the Departure Rate, the actual performance of a node, the average number of packets served in one second. Sometimes it is termed as "throughput", denoted as $\mathbf{D_i}$.

LossR. is the Loss Rate, the average number of dropped packets in one second, denoted as $\mathbf{L_i}$.

Util. is the Utilization of the node, the ratio of a node's throughput dividing its service rate, indicating the workload of this node, denoted as: $\mathbf{U_i = D_i/B_i}$

Link S. is the Link Speed, a measure of the output rate from a node to one of its neighbors. The link speed from node ***i*** to node ***j***, is denoted as $\mathbf{LS_{ij}}$. The departure rate of a node is the summary of its link speeds to all its neighbors:

$$D_i = \sum_{j=0}^{M-1} LS_{ij} = U_i \cdot B_i \qquad (2)$$

where M is the length of routing table on node ***i***. When $\mathbf{LS_{ij}} = \mathbf{LS_{ji}}$, this link is symmetric (otherwise asymmetric). The link speed and transport service time on this link can be displayed in the animation.

There are two array-counters counting the following two distributions:

- **Service time distribution** is the distribution of the service time of a node. It is counted since the simulation begins. With this distribution user can verify if this node's PDF option is working.

- **Arrivals/sec distribution** is the distribution of stochastic number of arrival packets per second, counted since the simulation begins. This distribution may take a certain Probability Mass Function (PMF, [19]).

Let's observe two counters on node M as an example: M is a CLASS_B router with multiple input and output routes: **X :=** 21ms, **Cap** := 53, **Dev** := 30.100 and **PDF** := GAUSSIAN. VINS plots the two counters in Figure 9 and 10.

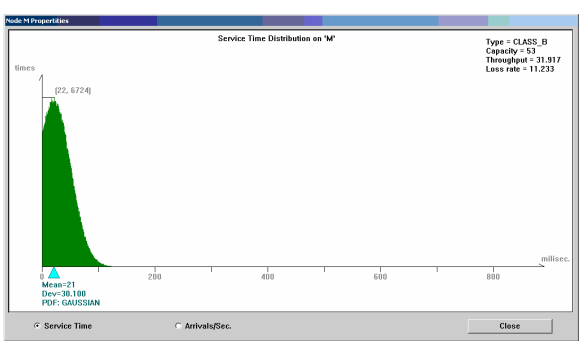


Figure 9 Service time Distribution on M

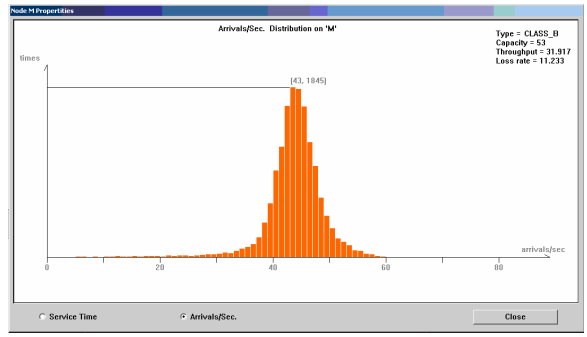


Figure 10 Arrivals/sec Distribution on M

As shown in figure 10: $\mathbf{A_M} = \mathbf{D_M}. + \mathbf{L_M}$ = 31.917 + 11.233 = 43.250 pkts/s (no packet is discarded on this node).

6.3 Flow and Network Statistics

To observe the flows' attributes in a complex network, counting and calculation are made as listed in the following:

Sent is the number of departure packets from the source (a SUPPLIER or application) since simulation started.

Rcvd is the number of arrival packets at the destination (a CONSUMER or application) since simulation started.

FL is the FLight-size, the number of packets of a flow that are being relayed in the network of a flow. FL indicates the network resource a flow temporarily possesses. It is also a desired variable in TCP congestion control computing, which can only be estimated in the real engineering.

SFL is the Smoothed FLight-size of a flow. As FL is stochastic that changes from observations, user can hardly use it to evaluate the size. Instead, SFL might be a useful measure to indicate the resource that a flow averagely possesses. The calculation of SFL is:

$$SFL = \frac{\sum_{i=0}^{N-1} (t_{i,e} - t_{i,s})}{E} \qquad (3)$$

where $(t_{i,e} - t_{i,s})$ is the lifetime of packet i since the creation to vanish (being received, dropped or discarded); E is the simulation's Elapsing time; N is the packets number of this flow that have been injected into the network. SFLs can be displayed with VINS' "Net-Pie Chart" (Figure 11).

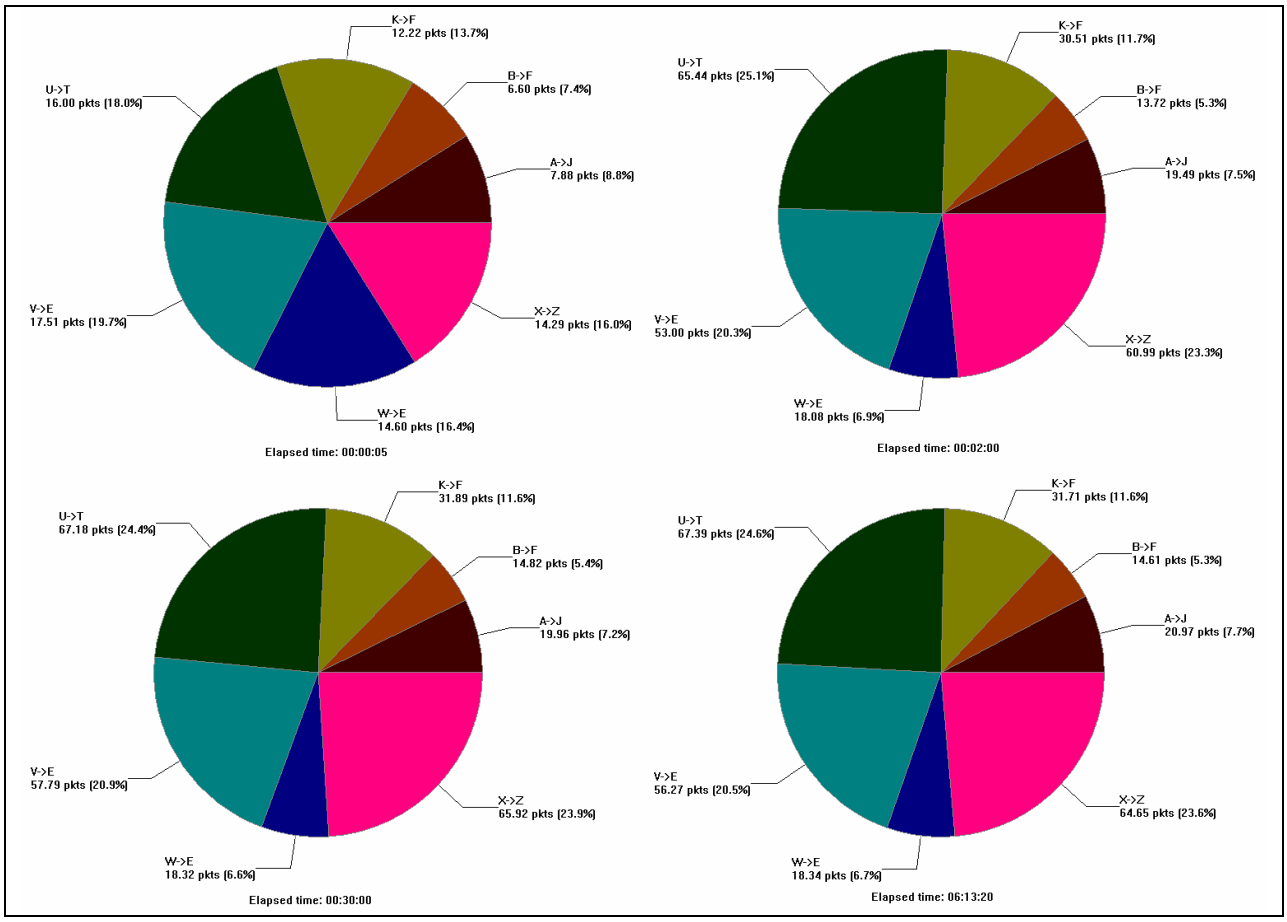


Figure 11 VINS Net-Pie Charts

This figure contains a series of Net-Pie charts that are generated after this scenario had ran for 5 seconds, 2 minutes, 30 minutes and over 6 hours. As presented, the SFLs are stabilized as the simulation time elapsing.

Reachability is Boolean type attribute of a flow. There are a lot of factors that may cause a flow to be un-reachable, such as destination not exist, detached, BGP mis-configuration [20], TTL expire and etc.

TP is the throughput on this flow, counting the arrival rate at the destination.

Delivery is the percentage that the number of successfully delivered packets is in the amount of all packets in this flow.

MOTT is the Mean One-Trip-Time, the average time that a successfully delivered packet takes (unit: millisecond).

7. Conclusion and Future Works

VINS is created based on a Node-oriented model for IP Network simulation. Some aims have been achieved: the model is distinctive and solid; its simulation core's structure is close to the implementation in the 4.4BSD kernel's; the inter-domain simulation issue has been thoroughly solved by adopting the Classful network addressing / subnetting scheme in building up scenarios; it maintains a big number of counters so that calculation can be made on node, flow and network levels; XML is chosen as the encoding language as its structure is simple, meaningful and flexible to encode complex information. It can be extended when new network properties are found to be important in the simulation.

The visualized simulation process of VINS may change the ways that people think and study IP networks back to the entities in the real world – the routers and hosts. It may help people get a valid picture in understanding the responsibilities and

issues in the IP Layer and run simulations which are closer to the real ones. It can be a useful tool in teaching, engineering and scientific researches.

The development is advancing in the Transport Layer. There is a list of targets to be achieved in a short future: 1) implementation of the application objects that can be mounted on HOST nodes and simulate Transport Layer protocols, especially the TCP, which is reliable, flow and congestion controlled; 2) refine simulation of the transport servers in the Physical Layer, providing more simulative options, such as error-rate; 3) encoding and simulating time events for dynamic networks where the topology and traffics vary with time. In a longer future, a full graphical scenario editor is expected.

8. Acknowledgment

I would thank Professor Werner Zorn for helping me understand the transport servers and transport service time in the Physical Layer and kind support in the development of VINS and writing this paper.

9. References

[1] Kevin Fall and Kannan Varadhan, *The ns Manual*, April 14, 2002.
NS-2 Home: http://www.isi.edu/nsnam/sn/

[2] F. Baumgartner, M. Scheidegger, T. Braun. *Enhancing Discrete Event Network Simulators with Analytical Network Cloud Models*. International Workshop on Inter-domain Performance and Simulation (IPS), Salzburg, Austria, February 20-21 2003, pp. 21-30

[3] Srinivasan Keshav. *Real: A network simulator*. Tech-. nical Report TR-88/472, University of California,. Berkeley, December 1988

[4] C. Zhu, O. W. W. Yang, J. Aweya, M. Oullette, and D. Y. Montuno. *A comparison of active queue management algorithms using the OPNET Modeler*. IEEE Communications Magazine, 40(6): 158-167, 2002.

[5] Roland Bless, Mark Doll, *Integration of the FreeBSD TCP/IP-Stack into the Discrete Event simulator OMNET++*. Proceedings of the 2004 Winter Simulation Conference R. G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

[6] Bless, R. 2002, January. *Using Realistic Internet Topology Data for Large Scale Network Simulations in OMNeT++*. In 2nd International OMNeT++ Workshop. Technical University Berlin, Germany.

[7] M. Haas and W. Zorn, Methodische Leistungsanalyse von Rechensystemen, R. Oldenbourg Verlag, ISBN 3-486-20779-2, 1995

[8] Gary R. Wright and W. Richard Stevens, *TCP/IP Illustrated Volume II - The Implementation*, 1994

[9] A. Knöpfel, B. Gröne, P. Tabeling, *Fundamental Modeling Concepts –Effective Communication of IT Systems,* John Wiley & Sons, Ltd, 2005

[10] Floyd, S., and Jacobson, V., *Random Early Detection gateways for Congestion Avoidance* V.1 N.4, August 1993, p. 397-413. Abstract.

[11] Jon Postel, *Internet Protocol (Classful Networks),* RFC 791, 1981

[12] V. Fuller, Classless Inter-Domain Routing, CIDR: an Address Assignment and Aggregation Strategy, RFC 1519, 1993

[13] Andrew S. Tanenbaum, *Computer Networks, Third Edition,* Prentice Hall PTR, 1996

[14] Elliotte Rusty Harold, The XML Bible, 2nd Edition, 2004
XML Home: http://www.w3.org/XML/

[15] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides, *Design Patterns*, 1994

[16] George Shepherd and Scot Wingo, *MFC Internals: Inside the Microsoft(c) Foundation Class Architecture,* May 16, 1996

[17] Box G.E.P, M.E. Muller, *A note on the generation of random normal deviates, 1958,* Annals Math. Stat, V. 29, pp. 610-611

[18] VINS download available: https://www-fgks.hpi.uni-potsdam.de/fileadmin/user_upload/Nanjun/VINS.zip

[19] Roy D. Yates and David J. Goodman, *Probability and Stochastic Processes – A friendly Introduction for Electrical and Computer Engineers,* John Wiley & Sons, Ltd, 1999

[20] R. Mahajan, D. Wetherall, and T. Anderson. *Understanding bgp misconfiguration*. In Proceedings of SIGCOMM, Pittsburgh, PA, August 2002.

10. Appendix: System Report

Start time: Thu 11 Jan 2007 19:03:18 GMT / Elapsed time: 00:33:24

Node Level

Name	Type	Address	PDF	Cap.	X	Dev	Put	Loss	Discard	Arr. R.	Serv. R.	Dept. R.	Loss R.	Util.	Link S.
A	SUPPLIER	12.34.56.5	GAUSSIAN	15	70	20.000	26330	0	--	13.139	14.286	13.139	0.000	91.90%	C: 13.14
B	SUPPLIER	12.34.56.2	GAUSSIAN	15	100	50.000	18605	0	--	9.284	10.000	9.284	0.000	92.80%	C: 9.28
C	CLASS_C	12.34.56.33	GAUSSIAN	25	50	42.500	34554	10356	0	22.410	20.000	17.243	5.168	86.20%	G: 6.93 H: 10.31
D	CLASS_C	12.34.77.1	GAUSSIAN	30	44	51.200	36672	59822	0	48.151	22.727	18.299	29.851	80.50%	G: 0.00 E: 12.89 F: 5.41 N: 0.00
E	CONSUMER	12.34.77.22	GAUSSIAN	30	36	23.400	25823	0	--	12.886	27.778	12.886	0.000	46.30%	--
F	CONSUMER	12.34.77.23	GAUSSIAN	30	23	10.900	10849	0	--	5.414	43.478	5.414	0.000	12.40%	--
G	CLASS_B	12.34.71.1	GAUSSIAN	36	31	13.400	45055	0	0	22.483	32.258	22.483	0.000	69.60%	C: 0.00 H: 0.00 D: 22.48 M: 0.00 R: 0.00
H	CLASS_C	12.34.86.12	GAUSSIAN	26	80	62.100	20654	13	0	10.313	12.500	10.306	0.006	82.40%	C: 0.00 G: 0.00 J: 10.31 I: 0.00
I	CONSUMER	12.34.86.96	GAUSSIAN	10	78	73.800	0	0	--	0.000	12.821	0.000	0.000	0.00%	--
J	CONSUMER	12.34.86.97	GAUSSIAN	21	76	20.900	20652	0	--	10.305	13.158	10.305	0.000	78.30%	--
K	SUPPLIER	12.156.11.97	UNIFORM	11	45	40.900	40224	0	--	20.072	22.222	20.072	0.000	90.30%	L: 20.07
L	CLASS_C	12.156.11.21	GAUSSIAN	41	36	10.900	49018	57573	0	53.189	27.778	24.460	28.729	88.00%	M: 24.46
M	CLASS_B	12.156.21.11	GAUSSIAN	53	21	30.100	67796	24098	0	45.855	47.619	33.830	12.025	71.00%	G: 7.09 N: 0.00 L: 0.00 R: 26.74 Y: 0.00
N	CLASS_C	12.156.199.11	GAUSSIAN	31	27	22.700	51469	0	0	25.683	37.037	25.683	0.000	69.30%	M: 0.00 P: 0.00 D: 25.68
P	CONSUMER	12.156.199.201	UNIFORM	11	17	26.170	0	0	--	0.000	58.824	0.000	0.000	0.00%	--
Q	CLASS_B	12.82.233.9	UNIFORM	48	27	24.400	58397	0	0	29.140	37.037	29.140	0.000	78.60%	R: 15.03 S: 14.11
R	CLASS_A	12.80.100.27	GAUSSIAN	78	37	34.400	45246	38373	0	41.726	27.027	22.578	19.148	83.50%	Q: 14.11 M: 0.00 G: 8.46
S	CLASS_C	12.82.103.12	GAUSSIAN	28	37	32.000	36531	31904	8926	38.603	27.027	18.229	15.920	67.40%	Q: 15.03 T: 3.20
T	CONSUMER	12.82.103.22	GAUSSIAN	34	56	43.000	6418	0	--	3.203	17.857	3.203	0.000	17.90%	--
U	SUPPLIER	12.156.11.96	GAUSSIAN	12	25	20.800	66408	0	--	33.138	40.000	33.138	0.000	82.80%	L: 33.14
V	SUPPLIER	12.82.103.24	GAUSSIAN	9	36	17.900	49103	0	--	24.502	27.778	24.502	0.000	88.20%	S: 24.50
W	SUPPLIER	12.156.199.202	GAUSSIAN	8	33	27.000	51470	0	--	25.684	30.303	25.684	0.000	84.70%	N: 25.68
X	SUPPLIER	12.156.22.191	GAUSSIAN	9	40	32.000	42999	0	--	21.457	25.000	21.457	0.000	85.80%	Y: 21.46
Y	CLASS_C	12.156.22.18	UNIFORM	23	39	40.000	42927	67	0	21.454	25.641	21.421	0.033	83.50%	M: 21.42
Z	CONSUMER	12.82.103.23	GAUSSIAN	14	65	41.000	0	0	--	0.000	15.385	0.000	0.000	0.00%	--

Flow Level

Src App	Route	Dest App	Proto	Mode	Sent	Rcvd	FL	SFL	Reach-ability	TP	Delivery	Mean OTT
--	A->C->H->J	--	--	--	26330	20652	40	21.04	Reachable	10.305	78.435%	2041
--	B->C->G->D->F	--	--	--	18605	5389	22	14.45	Reachable	2.689	28.965%	3094
--	K->L->M->G->D->F	--	--	--	40224	5460	42	31.31	Reachable	2.725	13.574%	4893
--	U->L->M->R->Q->S->T	--	--	--	66408	6418	93	67.35	Reachable	3.203	9.664%	8018
--	V->S->Q->R->G->D->E	--	--	--	49103	6530	59	55.23	Reachable	3.258	13.299%	6539
--	W->N->D->E	--	--	--	51470	19293	27	18.64	Reachable	9.627	37.484%	1768
--	X->Y->M->R->Q->S	--	--	--	42999	0	64	63.13	Unreachable	0.000	0.000%	--

Network Level

Input	Output	Livings	Throughput	Loss Rate
295139	63742	347	31.807	110.881

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Transformation of Existing Programs into Autonomic and Self-healing Entities

M. Muztaba Fuad[†] and Michael J. Oudshoorn
Department of Computer Science
Montana State University
Bozeman, MT 59717, USA
E-mail: {fuad, michael}@cs.montana.edu

Abstract

Autonomic computing is a grand challenge in computing, which aims to produce software that has the properties of self-configuration, self-healing, self-optimization and self-protection. Adding such autonomic properties into existing applications is immensely useful for redeploying them in an environment other than they were developed for. Such transformed applications can be redeployed in different dynamic environments without the user making changes to the application. However, creating such autonomic software entities is a significant challenge because of the amount of code transformation required. This paper presents feasible solutions to such code transformations and identifies associated issues. To illustrate such code transformations, a technique is presented that inserts Java byte code with self-healing primitives and transforms it to become a self-healing component. Experiments show that such code transformations are challenging however they are worthwhile in order to provide transparent autonomic behavior.

1. Introduction

It is not always a feasible option to build self-adaptive distributed systems from scratch. Mostly because of the cost and time associated with such a major development. Programming such a distributed application is also a tedious task and programmers need expert knowledge to handle the distribution management issues along with programming the problem at hand. For average programmers, this becomes a daunting task when they also have to incorporate autonomic primitives into the system. In real life, programmers' want to concentrate on the problem in hand, rather than spend time on incorporating autonomic behaviors in their system. It will be tremendously beneficial to programmers if such autonomic behaviors can be added automatically and transparently to existing systems. Although object oriented technology provides programmers with the advantage of rapid program development in a networked environment, it becomes overwhelming when it is necessary to handle the autonomic computing aspects of the program as well. Since program behavior is highly dynamic and a different distribution configuration may be appropriate in different phases of the execution of a program, components should be autonomic in behavior to increase productivity and system manageability. Such autonomic application can be redeployed in different dynamic environments without the user making changes to the application. However, there are many challenging aspects [1] associated with such program transformations and incorporation of autonomous behavior into existing programs. We envision such a distributed system as an Autonomic Computing [2] challenge where *Autonomic Elements* (AE) are incorporated automatically to handle the complexities associated with distribution, coordination and efficient execution of program components.

This paper presents techniques of injecting autonomicity into non-autonomous Java byte-code and transforming it into autonomous entities. Although the technique presented in this paper works with Java byte-code, any interpreted code such as C#, that utilizes standard object oriented primitives can utilize this approach with minor modifications. The existing code is analyzed and the autonomic primitives are inserted in such a manner that it is separated from the service functionality of the existing application. This paper identifies and presents the required transformations to make objects autonomic and discusses different aspects of such transformations. To illustrate code transformations, a technique is presented that transforms Java byte-code into a self-healing component. The

[†] *Partially supported by Montana NASA Grant Consortium, Award No. MSGC-426080/07*

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

approach presented in this paper transforms the existing user code in three phases. In the first phase, the code is analyzed and independent program partitions are identified. Please see [3] for more information on identifying such program partitions. Once partitions are identified, each partition is transformed into a managed element and housed inside a dedicated autonomic element. Parallel to these transformations, another phase of transformations adds self-healing primitives in the partitioned user code. The remainder of the paper is organized as follows: Section 2 identifies and lists required object level code transformations to make independent program partitions as autonomic entities. Section 3 presents the self-healing transformations that are added to the autonomic program partitions. Section 4 discuss about the experimented results. Section 5 presents similar works and finally Section 6 concludes this paper.

2. Required transformations and issues

This technique targets computationally intensive large parallel applications, which can be modeled as a collection of independent computing and communication resources with diverse capabilities into a large-scale integrated system. More about the underlying infrastructure and target application and generating independent partitions can be found in [3, 4]. Since the transformed program runs as a distributed program, any single node in the system could have one or more objects executing on it. As all of the related objects within any single node have to be manipulated transparently, several different scenarios appear for consideration:

i. *One AE per node with each object as a ME* (*Managed Element*): The traditional proxy approach [4] could be employed; however, the full computation power of a particular node is not utilized in this scenario. The traditional proxy approach only supports a single object per proxy, but we require multiple objects per proxy.
ii. *One AE per node with multiple objects as ME:* This is the most common scenario where multiple objects run concurrently in one node. The traditional proxy notation must be extended to address this scenario.
iii. *Multiple AE per node with multiple objects as a ME:* Although this scenario is possible, it is not supported by the autonomic computing paradigm and will not be considered.

A proxy structure is proposed that encapsulates one or more runtime objects into a single manageable entity that communicates with other such entities in the system with a single communication channel provided by the proxy object. Having an encapsulating proxy object allow us to incorporate the autonomic functionalities seamlessly into the user objects with the help of sensors, actuators and control interfaces [3]. Once user classes are transformed using the traditional proxy approach, a new proxy class is created to encapsulate all the other proxies in that node. Any inter-object communication between single ME proceeds as usual, however, any inter-object communication between two different MEs is delegated to the encapsulating proxy object. There are several possible choices to create such an encapsulating proxy:

i. *The proxy class inherits the original class:* This works for only one class as Java does not support multiple inheritance. This can be overcome by creating a new interface with all the methods in each of the target classes and then having the proxy implement that interface by copying the methods body into the proxy class.
ii. *Renaming the original class with the proxy class:* This is the traditional approach and does not work for the proposed approach as it is difficult to delegate all proxy invocations separately.

For such transformations of user code, the following issues need to be addressed:

1. <u>Methods (*M*) and constructors (*C*)</u>: There are more methods in the proxy class to interact with the AE and also to manipulate the objects itself. So, if M_n , $C_n \in$ Original Class $\rightarrow$ $M_{n'}$, $C_{n'} \in$ Proxy Class, where $n'>n$. The original methods are overridden with the following structure:

   Pre-processing
   Original method call
   Post-processing

   Instead of instrumenting each method, a wrapper method is created to ensure the consistency of the existing line number table for debugging purpose. Figure 1 shows a possible proxy creation scenario. The original object graph (Figure 1(a)) is transformed to a corresponding proxy object relationship. To learn more about generating object graphs and how to partition such graphs, see [3]. Next, each partition of the graph is encapsulated with a master proxy (Figure 1(b)) to add extra functionality and to have a second level of delegation. So, if there was a method call in U_1 as $X_2.foo(arg1,arg2)$, it is now transformed to $Proxy\_AE_1.\ invoke(X_2,\ foo,\ args)$, where *args* is an *Object* array having all the arguments of the calling method wrapped appropriately to their object classes. The *invoke* method inside $Proxy\_AE_1$ identifies in which node X_2 is located (node 2 for this example) and makes a remote call to the method $Proxy\_AE_2.invoke(X_2,foo,args)$. The remote invoke method unwraps the arguments and call the local method *foo* in X_2 and wraps any return value from that method and sends it to the caller ($Proxy\_AE_1$).

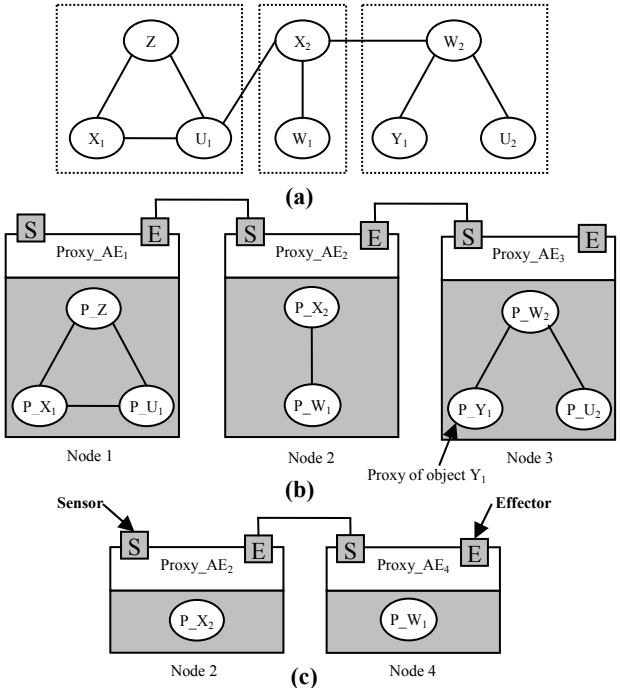


Figure 1. Object transformation.

Having an encapsulating proxy allows us to move an object from one node to another (Figure 1(c)) for optimization purposes by creating a new proxy wrapper in the new node that encapsulates the migrated object. This allows calling a method even if the original object has migrated to some other node and the original proxy is left somewhere else.

2. Polymorphic method calls: To determine the original calls of a method in a polymorphic call requires a fully fledged stack oriented emulator as in the JVM. Creating one such emulator is a separate research problem and in the initial version of the transformations, polymorphic method calls are not considered.
3. Direct field access: All direct field access is converted to *getter* and *setter* methods to facilitate remote method invocation.
4. System classes: Since the system classes cannot be modified, following the same techniques as J-Orchestra [6], system objects are either moved with the user objects or if they use any platform dependent code then remains in the same node and other proxies access those objects using callback facility.
5. Handling distributed I/O: It is undesirable to have user code produce output in a remote machine or ask for input somewhere other than it is intended to. Therefore all input/output operations need to be redirected. This leads to the following possible transformations:
   a. *Standard output and error*: The user has the choice of redirecting all remote output/error to the client machine (undesirable-may overwhelm the network), save it in a log file in the remote machine or ignore it as a whole.
   b. *Standard input*: All standard input is redirected to the client machine.
   c. *File input*: The user has the choice of redirecting all file read operation to the client machine or the underlying framework copy the file to the remote machine and file input operation proceed as usual. Both of these choices have their own tradeoff and one is better than the other in different situations.
   d. *File output*: File output could be redirected to the client machine or saved as a local file in the remote machine.
6. Exception handling: Section 3.7 illustrates the approach toward handling exceptions. Users could utilize different system level policies to steer the exception handling approach, such as permit the exception to propagate through the class hierarchy or try to heal the exception with user interaction or previously saved solutions.
7. Final class: To extend the functionality of a un-modifiable class in the proxy, the final keyword is removed from the *classpool* [7] inside the class file. This way, the semantic and functional consistency of the class remains the same but now extra methods can be added and existing methods can be overridden in the proxy to add the extra functionality.
8. Existing interfaces: Since Java allows a single class to implement as many interfaces as required, no changes are required in this case.
9. Static methods and fields: Any class that has static methods and fields, is divided into two subclasses where one has all the static methods and fields and other has the non-static entities. Separate proxies are created for each subclass and static sub class is anchored in one node and interacted by other proxies using RMI callbacks.
10. Use of *this* and *super*: Use of *super* does not cause any problem in the transformed code. However, the use of *this* need to be delegated to appropriate proxy class.
11. Use of reflection: Reflection in the user code is not considered due to the added complexity in the byte code rewriting phase. A separate package on reflection that delegates user enforced reflection must be developed to address this issue.

To handle other Java language features, the techniques used have similarities with the techniques used in J-Orchestra [6]. The major distinction with this approach is that we are now attempting object level

distribution, whereas J-Orchestra uses class-level distribution. One significant drawback with both of these approaches is that it incurs extra space and time to run the resulting distributed application. Since new proxy classes are created and segments of byte code are inserted into the exiting byte code, we expect to see some inflation in the resulting code size. However, we believe that, for the sake of the transparency and the added dynamics in the system, this added space penalty is worth taking.

Experiments were conducted to find the time requirement of delegated method call using proxy-based autonomic elements. Every remote method call now takes place through the autonomic element. Therefore the call can be broken down into several parts: 1) Call to local proxy 2) Call from local proxy to remote proxy (AE) 3) AE locates the remote class, loads it and uses runtime reflection to delegate the execution to the corresponding method. Figure 2 shows timing for such remote calls, where other denotes the time it takes for local proxy class loading and locating the remote AE using standard RMI naming protocol. For single remote call using the delegated proxy pattern, the increase in execution time is around 45%. With several calls to the same AE, this reduces to less then 10% as the JVM normally caches classes and for multiple calls, class loading and reflection occurs faster than the initial time.

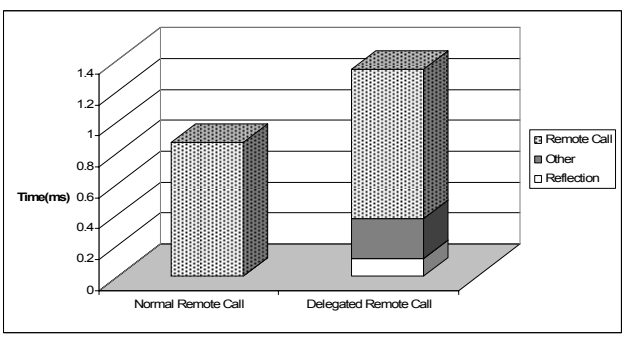


Figure 2. Timing for invocation through proxy.

3. Self-healing Transformations

Once the program partitions are transformed into independent autonomic entities by encapsulating them inside an autonomic element (Section 2), the next phase of transformations are required to add the different autonomic properties into those transformed partitions. This section presents an approach to incorporate self-healing primitives inside existing user code once the partitions are transformed into autonomous managed elements. An autonomic self healing application should be able to recover from faults and should continue to work smoothly [2]. The traditional approach for having such self-healing capabilities in an application is to build the application in such a way to have those functionalities built into it (hardwired). This section presents a technique of code transformation and injection to automatically transform existing programs into self-healing entities. The following sub-sections describe this technique in more details.

3.1 Faults and fault model

The approach presented in this paper is concerned with faults that occur after the program is deployed. Such faults could result from problems or bugs in the user code, in the underlying physical node or in the run-time environment. Faults caused by bugs in the user code (logical errors), user generated custom exceptions or faults generated due to the functional aspect of the program is outside the control of this approach and should be addressed by the system administrator or the developer of the user program. A fault can be either transient (network outage, memory overload, disk space outage etc.) or non-transient (bug in the user code) [8]. The technique presented in this paper can self-heal such transient faults as once the system restarts after self-healing, the condition that caused the fault will be healed and will not reappear. However, non-transient faults are generally caused by some bug in the code or due to unhandled exceptions. Although some types of non-transient faults could be self-healed, this may change the original semantics of the program or the fault will occur again as the condition that is creating the fault actually resides in the user code.

Each recoverable fault is represented internally by a four tuple $<F_{name}, F_{index}, F_{stack}, O_{info}>$ fault model, where,

F_{name} = Name and type of the fault.

F_{index} = Byte code index where the fault occurs. This must be deduced from the byte code line number attribute [7] inside the class file as the JVM only gives the source code line when throwing an exception.

F_{stack} = Stack trace of the exception, transformed into chained fault models.

O_{info} = Object information such as, object and method name and formal and actual parameters of the method that caused the exception.

This representation allows the system to manage faults more easily and effectively. Representing faults in an accurate and consistent manner is not enough; the associated actions also have to be represented in some open standards so that the model can be extended or updated without the need to modify the underlying algorithm. There can be multiple possible actions for any certain fault. The number of such possibilities grows or reduces over time as the system adapts with feedback

```
<EQUAL>
  <IOException>                                  Pre-condition
    <FileNotFoundException>
      <RESULT=CALL doSearch (PATH, filename) />
        <STRINGCONSTANT>
          <PATH=<VALUE>CLASSPATH</VALUE> />      Action and
          <PATH=<VALUE>PARENT</VALUE> />         associated
           <PATH=<VALUE>ROOT</VALUE> />          setup
        </STRINGCONSTANT>
        <PRIORITY> 5 </PRIORITY>                 Priority
      </RESULT>
      <STRINGCONSTANT>
        <CLASSPATH=<VALUE>CLASSPATH ADD RESULT</VALUE> />
      </STRINGCONSTANT>
    </FileNotFoundException>
      ....                                       Post-condition
  ....
  ....
<NOT>
  ....
  ....
```

Figure 3. Representation of fault model.

from the system administrator or through learning by reinforcement ($Action_{xn}$ did not heal $Fault_x$ for n number of times, so reduce its priority or disable the action altogether). We can deploy different statistical and machine learning algorithms to learn new fault scenarios; however, it is out of scope of this paper. Autonomic Computing Policy Language (ACPL) [9] is adopted to describe policies of different fault scenarios. This not only allows us to address a consistent policy management across the system [3], but also provides a user friendly form of policy definition, policy management and tools and API to work with policies. The fault repository represents all fault information and action using ACPL. Figure 3 gives a snapshot of the fault repository database that stores all fault models. All faults are described by a pre-condition (what to match), one or more actions with associated setup, priority of a certain pre-condition and any post-conditions. Priority is used to select among multiple actions for the same pre-condition. Assignment of priority to an action is a challenging issue and a simple utility based priority management algorithm [10] is being used to assign priorities among multiple actions. Since ACPL is XML based, future extension of this technique or integration of this technique into an autonomic environment would be simpler for the application programmer.

3.2 Transformations

Byte code rewriting tool Javassist [11] is used to introspect and retrofit existing byte codes with self healing primitives. To handle any runtime exceptions, an extra try-catch block is inserted in every method. At runtime, if any exception occurs, it is caught and analyzed to find the cause of the exception.

All analyzed information is saved as logs, so that the system administrator can later inspect and take proper actions. Once such actions are taken, it is recorded to the fault repository so that next time the same fault can be avoided. A clone of the running method is created where, at the beginning of the method, new code is inserted to reinitialize the object with a consistent state and to restart the method at the point where the fault occurs. Since the rest of the code already has the injected self-healing mechanism, any further faults are handled in the same way as described above. Several adaptations are needed to support existing programs with the desired autonomic behavior. The transformations that take place during the lifetime of a method are as follows:

1. The original method body is wrapped in an extra *try-catch* block and checkpoint calls are inserted in the body itself.
2. If any exception is thrown by the code, it is caught by the newly inserted *try-catch* block.
3. The caught exception is analyzed and the process starts to heal that particular fault.

3.3 Local variables

To save the state information of any running method inside the object, local variable values have to be saved during checkpointing along with other necessary information, such as the line number of the next instruction to be executed, value of actual parameters and value of any object field. To gather the current state of the object, all the local variables in the current scope along with any fields and class variable's values need to be saved. As the Java compiler does not save any local variable information (such as type information, variable scope etc.), the static analyzer gathers this information by analyzing the byte code and adds that information as a byte code attribute [7] to each method. During self-healing transformations, this attribute is used to create appropriate wrappers to save the corresponding object status. The challenge is to recreate a local variable table from the byte code when there is no information saved for the local variables. Byte code instructions access local variables from a zero based array in the runtime stack and depending on the type of method (class or instance), variables and arguments are indexed differently.

Along with all local variables, method parameters and object instance (only with instance methods) are also placed on that array. Depending on the data type, variables will occupy 1 or 2 cells in the array. Depending on the scope of the variable, cells could be reused by different variables within different scopes of the method body. The challenge is to represent such dual usage in a consistent manner so that the local variable can be accessed with a proper type cast at the proper location in the byte code. The algorithm adds a duplicate entry with the same index into the local variable table for

such occurrences. Different scope information is used to distinguish between these two entries.

To determine variable scope, variable index within the array and data type, the following algorithm is devised:

1. *For all byte code instructions in a method, do*
2. *Get next opcode (and operand, if any)*
   a. *If the opcode is of a data store type, then determine the type from the opcode (for primitive type). For reference (object) type, check the constant pool and determine the type information.*
   b. *Determine the variable index from the opcode suffix or from the operand. Also, determine the corresponding byte code offset, where it is declared for the first time.*
   c. *If the opcode represents an already identified variable with the same data type, then update that variable's scope information.*
   d. *If the opcode represents an already identified variable with different data type (same variable index is used by two variables in two different scope), then create a new variable entry.*

Algorithm 1. Generate local variable information.

The time taken by algorithm 1 is proportional to the number of local variables in the method. Only 8 bytes are used to save all necessary information about a local variable. This keeps code inflation to a minimum. See Section 4 for some comparisons. The algorithm requires only one pass over the code to determine the local variables and it is performed simultaneously during the static analysis [3], therefore there is no extra overhead for executing this algorithm.

3.4 Adding checkpoints

To restart seamlessly from a crash or after a runtime exception, the internal state (fields, method parameters and local variables, next byte code to be executed) of the object has to be made persistent. According to the given user policy, instrumentation points in the code are determined and appropriate method calls are inserted to checkpoint the current state of the object. The following checkpoint locations are identified at byte code level:

i. After several write operations.
ii. Before and after an I/O interaction.
iii. Before and after any data interaction between two objects.

The checkpointing algorithm described below adds a status object as a field into the target class, so that the checkpointed data can be saved through that field. Algorithm 2 for checkpointing in byte code is used on a per method basis and uses different byte code attributes, such as code, exception, line number and local variable (generated by algorithm 1) attribute [7].

Proper type casting is necessary to store and to restore checkpointed data. All the local variable and actual parameter information is type sensitive and

1. *Insert an array of objects at the top of the method body. The size of this array is equal to the number of local variables in the method.*
2. *Determine the next checkpointable position in the byte code.*
3. *Assign object array with local variables having corresponding index value.*
4. *Load this object array and all other status information in to the run time stack.*
5. *Call the checkpoint method, which takes all the status information from the runtime stack and saves that in a persistent store.*

Algorithm 2. Checkpoint a method at the byte code level.

requires careful type manipulation. All primitive method data is converted to its corresponding object format and saved as objects in the persistent data store. During restoration, data is cast back to its original type by gathering the type information of each local variable and actual parameter from the local variable attribute.

3.5 Status data

During runtime, whenever any checkpoint is encountered, the current status of the object is incrementally saved to a platform independent disk file. The implementation of the checkpointed data manipulation algorithm is performed by synthesizing and re-engineering ideas found in existing fault tolerant techniques for distributed applications. Using pre-defined policies, users could modify the behaviour of such algorithms to suit their needs. For example, to save disk space, once an object quits its execution and is garbage collected, all the corresponding status information related to that object are deleted from the status data file. After a fault, determining a consistent state from where the program should be restarted is complicated and requires extensive analysis of the runtime structure. However, to keep it simple, we are only considering the last checkpointed consistent state to resume execution.

3.6 Fault analysis and fault healing

The crucial part of the self-healing is performed by the fault analyser-fault healer pair. The fault analyser code is inserted in the byte code level, where as the fault healer is a separate class structure that is initiated by the fault analyser. The fault analyser code inserted inside the method body is responsible to gather the necessary status information and to generate code to create the failure information and then call the appropriate algorithm to analyse the failure information and compare that with the saved fault models to produce a plan of action for the fault healer. Once the fault healer gets the required

healing information, it performs the following steps to resume the operation:

i. Find the last consistent state (a consistent state is where all the threads were able to write their status successfully) and read the saved status data from the backing store. The backing store is implemented hierarchically across the network and after a fault, if the backing store in the application's current domain is not available, then the backing store in its parent's domain is accessed and this propagates in worst case to the top of the hierarchical domain based backing store.
ii. Create a clone of the method.
iii. Inject byte code at the top of the method that reinitializes the internal data structures of the object with the status data. The last instruction in this newly added sequence is a *goto* instruction that directs the execution at the position in the method where we want the execution to resume.
iv. Start executing this cloned method using reflection and on the same object instance as the original method was running.

Normally, the fault healer saves all intermediate and cloned methods and associated byte code in the hard drive for debugging purposes. However, if the system administrator thinks that unnecessary, then it can be switched off and that will not only save disk space but also will speed up the healing process.

3.7 Runtime exceptions

To handle any runtime exceptions, the method body is encapsulated by another *try-catch* block (which catches the Throwable super class) to give the method another opportunity to continue after the statement where the exception is thrown. However, adding a new try-catch block introduces different scenarios which need to be addressed at byte-code level to fully support runtime exceptions:

i. *Subclass of Throwable or Exception is already being caught by the method body.*

   Solution: Re-throw that exception so that the enclosing try-catch block can catch it and continue with its self-healing mechanism.

ii. *There is already an encapsulating try-catch block that catches either java.lang.Exception or Throwable.*

   Solution: Instead of adding a new try-catch to enclose the entire method, the catch block is modified and self-healing mechanism is added.

iii. *A nested try-catch block already exists within the method body.*

   Solution: Re-throw every nested exception that is caught, so that the outer most enclosing try-catch block can catch the exception and continue with its self healing mechanism.

Since the rest of the code already has the injected exception handling mechanism, any recurring faults will be handled same way as described above. If the same fault is happening over and over again, after few tries, the system administrator will be notified to take proper action and the system will stop running that particular program.

4. Performance Analysis

To judge the applicability of this technique in a real environment, traditional benchmarking techniques are not enough. Simply measuring how fast a fault can be healed is dependent on the underlying platform and the runtime environment. Healing the same fault in two different platforms will take different times due to different environment parameters. Providing only such functional evaluation of the technique is insufficient to judge its effectiveness in the field of autonomic computing. Since one of the main goals is to provide a transparent interface to the user, an evaluation framework is developed to address such issues that can not be functionally evaluated. Table 1 shows such an evaluation of this technique. As noted in [12], benchmarking an autonomic system is difficult and there is no good way to do that. Since there are no other similar works to compare this technique with, we measure the time taken to do the transformations, the resulting code inflation and the runtime memory usage inflation.

Experiments on the presented self-healing techniques were conducted with several Java programs (sequential and parallel) running in a single machine, having varying code structures and complexities. Program 1 tries to write different data types on an open socket to simulate network failure scenarios. Program 2 writes random bytes on disk files to simulate I/O failures. Finally program 3 implements a concurrent matrix multiplication algorithm to simulate synchronization and locking faults. The techniques described in this paper can support Java byte code from JDK version 1.2 to 1.5. A standard PC equipped with a Pentium 4, 3.4 GHz CPU and 512 MB RAM running Windows XP is used to conduct all the experiments. To generate the local variable information, algorithm 1 produces smaller files than using the Java compiler's debugging switch (javac -g). Since programmers may not have compiled the original code using that switch, the self-healing code injector must rely on algorithm 1, which, as shown in Table 2 is worthwhile to use.

Table 1. Evaluation framework for our technique.

Properties	Abilities of this technique
Usability	This technique is easy to use and programmers do not have to worry about pre and post processing.
Modularity	This technique is designed in a modular fashion so that it can be added or deleted from an existing system without any loss of original program semantic.
Transparency	It is completely transparent to the user and using this technique is straightforward.
Scalability	This technique adds self-healing primitives on a per method basis. So other autonomic properties can be added in the same manner without the need to modify this technique. However, with any such code injection, programmers have to be aware of the resulting code inflation.
Persistence	This technique does not modify any internal data of the program. Hence the program execution remains unchanged functionally and semantically. Checkpointed status data is kept in a machine independent format using Java Serialization and saved in different portions of the data store to keep it consistent between runs.
Maintainability	Over a long period of time, the system could generate valuable usage data and execution pattern that will help software operators and programmers to maintain the code more effectively.

We experimented with two different forms of checkpointing policies. In the conservative policy, checkpoints are inserted after every statement in the method, whereas in the optimized policy, we used checkpointing decisions as described in Section 3.4. The resultant code has a *O(n)* code inflation, where *n* is the size of the original code. Figure 4 shows the overall code inflation due to the added functionalities in the code. On average we have around 51% code inflation. With better optimization techniques, this could certainly be lowered.

Figure 5 shows the timing data taken during execution. It is evident from Figure 5 that the algorithms take linear time to execute. For the added functionalities in the code, we have to be comfortable with such a small increase of pre-processing time along with the running time. We also measure the amount of memory the added data structure takes during runtime. Just for this test, we instrumented the code in such a way that it keeps track of the amount of heap memory available to the JVM at each step of the self-healing process. On average, the runtime memory inflation is about 10% to 15%. This is a minimal increase of runtime memory usage that will have any significant adverse effect on the program execution.

Table 2. Code Inflation due to local variable addition.

Program	Original Size	Algorithm 1	javac –g
1	910 byte	1036 byte	1228 byte
2	1208 byte	1377 byte	1639 byte
3	5823 byte	6155 byte	7320 byte

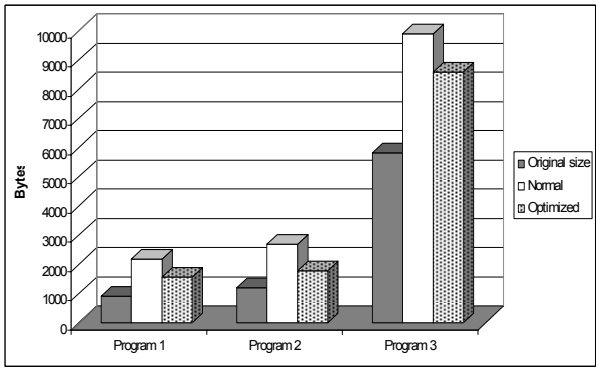


Figure 4. Code Inflation due to self-healing primitives.

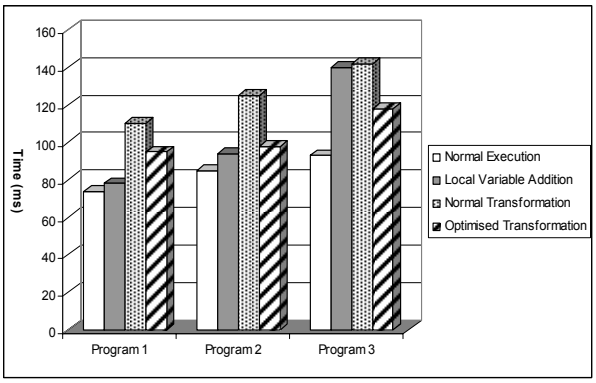


Figure 5. Execution times.

5. Existing works

Relatively little research has been done to integrate autonomic functionalities into existing code and most of them do not share the same goals as of this research.

Addistant [4] is a Java byte code translator for automatic distribution of legacy Java software. It takes Java software to be partitioned and uses a separate user specified placement policy to translate it to a distributed version. Addistant requires placement policies be specified at the class level, limiting the opportunity to exploit object-level concurrency.

J-Orchestra [6] is an automatic partitioning system for Java programs. J-Orchestra operates at the byte-code level and rewrites the application code and replaces local data exchange with remote communication (e.g., Java RMI, indirect pointers). It uses information from static

analysis and execution profiling to undertake the partition decisions. It requires no annotative inputs from users other than the network locations of various hardware and software resources. J-Orchestra also operates at class level granularity but unlike Addistant it does not require the user to explicitly specify policy for every class, instead it includes automatic analysis that ensures the correctness of partitioning.

Although the approach presented in this paper uses some of the ideas and techniques used to implement J-Orchestra, we present the transformations from the autonomic programmer point of view to program a distributed system full of concurrent objects. Our goal is to transform any objects into autonomic entities by transforming them at byte-code level, where as J-Orchestra transforms objects only to be run in a distributed fashion. J-Orchestra also uses intermediate source level translation, which incurs additional compilation time for conversion to byte-code. Our approach does all transformation in byte-code level, so there are no overhead related to compilation. J-Orchestra provides partial fault tolerance, on the contrary, self-healing is a major aspect of our transformation approach.

The work by Schanne *et al.* [14] attempts to inject autonomic functionality to existing object oriented code. The authors used the standard proxy/wrapper architecture to inject the added functionality, namely self-updating, self-configuration and self-optimization. One of the major drawbacks of their approach is that user needs to supply the pre-processor some meta information about the code, such as, pre and post-conditions and invariants of methods. So the assumption is that the user knows about the code and could express such meta information about the methods. This assumption is not realistic for legacy code, as exploring the meta information for such code is nearly impossible.

Traditional software fault-tolerance in Java is being actively studied by a number of researchers [15 - 27] in different aspects of software development. However, these approaches either work towards traditional hard coded fault tolerance or have different goals as this work:

- Operate at the source code level [15].
- Program through pre-defined interfaces to utilize such approaches [16].
- Deal with runtime software component update [17, 18].
- Employ serialization techniques, which places extra burden on the programmers by requiring them to implement the serialization and de-serialization methods [19].
- Use monitoring and profiling during run time [20, 21].
- Use a modified Java Virtual Machine for better control over the checkpointing and recovery process [22, 23], however it sacrifices portability and interoperability for doing so.
- Use different programming languages [24, 25, 26].

The technique presented in this paper differs from the above works and from traditional fault tolerance approaches in the way that it relieves the application programmer from the burden of complex programming interfaces and ever changing metaphors by abstracting all such transformations automatically. Also, programmers do not have to learn any new programming interface to utilize this technique. We believe that, any autonomic and self-managed system should be easy for the users to program, operate and maintain; otherwise the goal [2] of autonomic computing is sacrificed. Having the same goals as this work, relatively little research has been done to integrate self-healing functionalities into existing user code and, typically, the options explored are not transparent to the user and, in fact, require the user to have an extended knowledge of the code.

Haydarlou *et al.* [27] presents an approach and a conceptual architecture for fault diagnosis and self-healing of interpreted object oriented application. Although this work shares similar goals and uses a similar approach, the solution provided by this paper is extensive and completely transparent to the user. Moreover, this paper provides details of code injection methodologies and address different issues that arise during such code injection. The authors experiment with their proposed technique with an application, for which there is no need to save any state information for a restart after a fault. Therefore, their approach is not well justified in the case of real-world programs, having nested try-catch block, local variable interactions, conditional branches, etc. The authors also do not provide any code inflation or execution time information for their proposed approach.

6. Conclusions

Autonomic computing is a new paradigm where computing systems possess the capability of self-management. Building an autonomic distributed system from scratch is desirable however is not always an option. Mostly because of the cost and time associated with a major development like that. It will be tremendously beneficial to programmers if such autonomic behavior can be added automatically and transparently into existing code. This paper presents an approach to transform existing object oriented programs

into self-managed programs. This technique transforms user objects into autonomic entities and injects the existing user code with autonomic capabilities by statically analyzing the code and instrumenting it in such a way that the application components become self-manageable components. To illustrate that such transformation is possible; a technique to make existing object oriented programs self-healing is presented and evaluated. The goal of the code transformation is to make the transition as simple and easy to the application programmer in fulfillment to the goals of an autonomic computing system.

7. References

[1] Oudshoorn M. J., Fuad M. M. and Deb D., "Towards an Automatic Distribution System - issues and challenges", *The Int. Conf. on Parallel and Distributed Computing and Networks*, Austria, pp. 399-404, 2005.

[2] Kephart J. O. and Chess D. M., "The Vision of Autonomic Computing", *Computer*, Vol. 36, No. 1, pp. 41–52, 2003.

[3] Deb D., Fuad M. M. and Oudshoorn, M. J., "Towards Autonomic Distribution of Existing Object Oriented Programs", *Int. Conf. on Autonomic and Autonomous Systems*, IEEE Press, California, USA, pp. 17-23, 2006.

[4] Fuad M. M. and Oudshoorn M. J., "An Autonomic Architecture for Legacy Systems", *Third IEEE Workshop on Engineering of Autonomic Systems*, Maryland, USA, pp. 79-88, 2006.

[5] Tatsubori, M., Sasaki T., Chiba S. and Itano L., "A Bytecode Translator for Distributed Execution of Legacy Java Software", *ECOOP 2001*, Budapest, Hungary, pp. 236-255, 2001.

[6] Tilevich E. and Smaragdakis Y., "J-Orchestra: Automatic Java Application Partitioning", *ECOOP 2002*, Malaga, LNCS, Vol. 2374, pp.178-204, 2002.

[7] Sun Micro Systems, *Java Virtual Machine Specification*, 1999.

[8] Zhongtang C., Vibhore K. et al., "Utility-Driven Availability-Management in Enterprise-Scale Information Flows", *Technical Report*, College of Computing, Georgia Institute of Technology, USA, 2006.

[9] Dakshi. A., Kang W. L. and Jorge L., "Policy-Based Management of Networked Computing Systems", *IEEE Communications Magazine*, Vol. 43, No. 10, pp. 69-75. October 2005.

[10] Deb D. and Oudshoorn, M. J., "On Utility Driven Deployment in a Distributed Environment", *Fourth IEEE Workshop on Engineering of Autonomic Systems* (EASe 2007), Tucson, Arizona, USA, March 26-29, 2007, Accepted for publication.

[11] Chiba, S. and Nishizawa M., "An Easy-to-use Toolkit for Efficient Java Bytecode Translators" *Proc. of 2nd Int'l Conf. on Generative Programming and Component Engineering, LNCS 2830*, pp. 364-376, 2003.

[12] Aaron B. B. et al., "Benchmarking Autonomic Capabilities: Promises and Pitfalls", *First Int. Conf. on Autonomic Computing*, pp. 266-267, 2004.

[13] Serugendo G. D. M. and Deriaz M., "Specification-Carrying Code for Self-managed Systems", *IFIP/IEEE Int. Workshop on Self-Managed Systems & Services (SelfMan) 2005,* Nice, France, pp. 209-217, 2005.

[14] Schanne M., Gelhausen T., and Tichy W. F., "Adding Autonomic Functionality to Object-Oriented Applications", *14th Int. Workshop on Database and Expert Systems Applications*, pp. 725-730, 2003.

[15] H. T. Feng and E. A. Lee, "Incremental Checkpointing with Application to Distributed Discrete Event Simulation", *Technical Report*, EECS Department, University of California, Berkeley, April, 2006.

[16] Julia L. L. and Gilles M., "Efficient Incremental Checkpointing of Java Programs", *Int. Conf. on Dependable Systems and Networks (DSN 2000)*, pp. 61-70, 2000.

[17] Peyman O., Nenad M. and Richard N. T., "Architecture-Based Runtime Software Evolution", *20th Int. Conf. on Software Engineering (ICSE'98)*, pp. 177 -186, 1998.

[18] M. E. Segal and O. Frieder, "Dynamic Program Updating: A Software Maintenance Technique for Minimizing Software Downtime", *Journal of Software Maintenance*, Vol. 1, No. 1, pp. 59-79, 1989.

[19] Teha B. et al., "Implementing Simple Replication Protocols using CORBA Portable Interceptors and Java Serialization", *Int. Conf. on Dependable Systems and Networks (DSN'04)*, pp. 549-554, 2004.

[20] Kaiser G., Parekh J., Gross P., Valetto G., "Kinesthetics eXtreme: An External Infrastructure for Monitoring Distributed Legacy Systems", *Fifth Annual Int. Workshop on Active Middleware Services*, pp. 22-27, 2003.

[21] Rick K. et al., "The Recovery of Runtime Architectures", *news@sei*, Vol. 2, pp. 1-5, 2004.

[22] Tobias R. and Jesper A., "Dynamic Deployment of Java Applications", *Proocedings of Java for Embedded Systems*, London, May 2000.

[23] Kaffe VM for Java, http://www.kaffe.org/, 2006.

[24] Liu H. and Parashar M., "Accord: A Programming Framework for Autonomic Applications", *IEEE Transactions on Systems, Man and Cybernetics*, *Special Issue on Engineering Autonomic Systems*, Editors: R. Sterritt and T. Bapty, IEEE Press, 2005.

[25] Abbas N., Palankar M., Tambe S. and Cook J. E., "Infrastructure for Making Legacy Systems Self-managed", *Technical Report,* 2004.

[26] Griffith R. and Kaiser G., "Adding Self-healing Capabilities to the Common Language Runtime", *Computer Science Technical Report*, Columbia University, 2005.

[27] Haydarlou A.R., Overeinder B.J., and Brazier F.M.T., "A Self-healing Approach for Object-Oriented Applications", *3rd Int. Workshop on Self-Adaptive and Autonomic Computing Systems*, pp. 191-195, 2005.

Methods of Sensors Localization in Wireless Sensor Networks

Zenon Chaczko[1], Ryszard Klempous[2], Jan Nikodem[2], Michal Nikodem[2]
zenon.chaczko@uts.edu.au, {ryszard.klempous, jan.nikodem}@pwr.wroc.pl,
[1]University of Technology Sydney, *PO Box 123, Broadway NSW 2007, Australia*
[2]*Wroclaw University of Technology, Wybrzeże Wyspiańskiego 27, 50-370 Wroclaw, Poland*

Abstract

In recent years there has been a growing interest in wireless sensor networks (WSN) applications. Such sensor networks can be used to control temperature, humidity, contamination, pollution etc. Self-organization and routing algorithms dedicated to wireless sensor networks usually assume that sensors absolute positions are unknown and all decisions are based on sensor's own local information. This assumption makes wireless sensor networks more flexible and energy conserve because making decisions locally is faster and energy efficient. But sooner or later sensors positions have to be found (when sensor sends a message about some event we of course would like to know where this event takes place). In this paper we describe different solutions of finding transceivers positions in wireless networks and we discuss localization in wireless sensor networks. We propose to transfer localization function from base stations to every sensor. We evaluate presented method using simulations.

1. Introduction

In recent years there has been a growing interest in wireless sensor networks applications [3,10,16]. In this paper we use term wireless sensor network to refer to hundreds or thousands sensors that are deployed over some area. They can be used in hostile environments, i.e. battlefields, vast forests, fire endangered or security areas. Sensors are deployed e.g. using plane, so we can assume that their positions are random and unknown [4,5,20]. Protocols dedicated to WSN allow such structures to work properly without knowing sensors' absolute positions but this information is needed if we have to locate reported events (which is an aim in most of applications). We assume that because of cost and power consumption sensors are not equipped with GPS devices [12]. That is why we have to find algorithms of finding sensors' positions in the area [1,14,16].

At the beginning we describe methods of estimating positions in wireless networks. In third section we discuss sensors' localization in wireless sensor networks. In fourth and fifth section the way to minimize positions estimation errors is described and simulations results are presented.

2. Methods of localization

We can find transceivers' position on 2D area using three methods:

- by knowing a distance and a direction from one point *(x,y),* where both *x* and *y* are known,
- by knowing direction from two points (x_1,y_1) and (x_2,y_2) where x_1, y_1, x_2 and y_2 are known (AoA algorithm),
- by knowing distances from three points (x_1,y_1), (x_2,y_2) and (x_3,y_3) where x_1, y_1, x_2, y_2, x_3 and y_3 are known (ToA, TDoA, RSS algorithms),

In first and second approach it would be necessary to use directional antenna. In Angle of Arrival (AoA) algorithm [6,7,15,18], presented on Fig.1, transmitter P in position *(x,y)* sends a signal. Directional antennas in points (x_1,y_1) and (x_2,y_2) detect angles α_1 and α_2. Intersection of lines described by known points and angles determines where transmitter *P* lies.

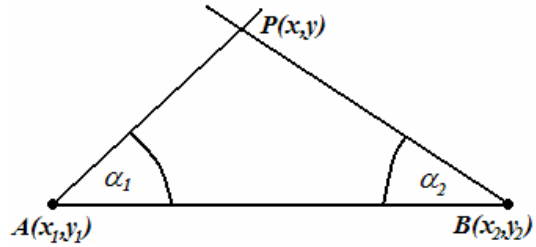


Fig.1. Localization using AoA algorithm.

Unfortunately requirement of using directional antennas makes these method very impractical for wireless sensor network application. Third method that bases only on estimating distances between nodes, is more interesting.

If we have three known points $A(x_1,y_1)$, $B(x_2,y_2)$, $C(x_3,y_3)$ and we know that point $P(x,y)$ (which position we are looking for) is distant d_1, d_2, d_3 from these points respectively, we can find *x* and *y* as is presented on fig.2.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

There are two exceptions. First when at least two known points have the same position. Second when all three points A, B and C lies in one line (see fig.3). In other cases unknown position of P can be found using three points A, B and C. We can easily see that the problem of finding transceivers' P positions is reduced to finding distances between pairs of transceivers (AP),(BP),(CP).

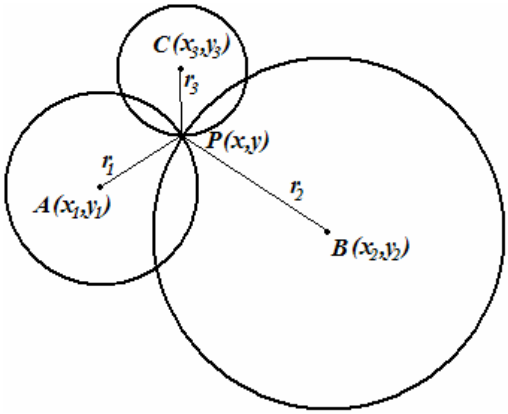


Fig.2. Finding point's position knowing distance to three known points.

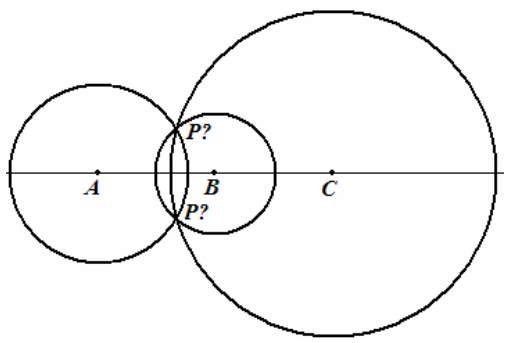


Fig.3. When known points A, B and C lie in line point P cannot be determined uniquely.

Methods of estimating distances are generally related to circumstances of signal propagation. Two main parameters are attenuation and delay. If signal with power P_0 is sent in a moment t_0, then its power when received in a distant place equals:

$$P_1 = \frac{P_0}{d_1^{\,m}} \tag{1}$$

and is delayed

$$t_1 = \frac{d_1}{v}. \tag{2}$$

In equation (1) and (2) d_1 is a distance between two points, m is a constant that depends on environment and v is a speed of a signal propagation in this environment. As we see both parameters can be used to estimate distance d_1.

Time of Arrival (ToA) [6,8,13,18] algorithm bases on signal delay. If sensor P with unknown position (x,y) sends signal $s(t)$ the received signals are

$$y_j(t) = k \cdot s(t - t_j)\,, \tag{3}$$

where j = 1,2,3 refer to receivers located in known positions (x_j,y_j). Assuming a perfect synchronization distances between transmitter P and receivers can be easily computed. In the other case we are forced to use Time Difference of Arrival (TDoA) algorithm [7,9]. When transmitter P sends a message then analyzing received signal correlation (delay) in more than two receivers we can compute distances between point P (which position we are looking for) and each receiver (which positions are known).

Unlike ToA and TDoA, RSS (Receive Signal Strength) algorithm bases not on signal delay but on signal strength analysis [2,13]. If the transmitted signal strength P_0 is known, distance between sensors can be estimated by measuring received signal strength P_1 as follows:

$$d_1 = \sqrt[m]{\frac{P_0}{P_1}}\,. \tag{4}$$

We assume that signal strength decreases with the distance. Disadvantage of this method is necessity of estimating constant m that depends on environment conditions.

On fig.2 we present that by knowing distance to three different known points A, B and C we can easily compute common point P no matter which method was used to find these distances. But what happens if we consider some error in estimating distances r_1, r_2 and r_3? Fig.4 shows situation similar to one from fig.2 but because of an error in distances estimation three circles will not have a common point.

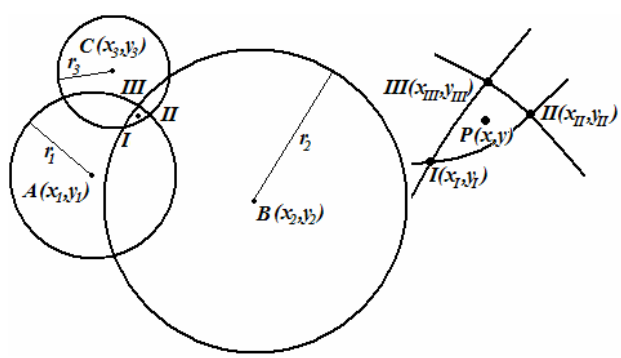


Fig.4. Finding point's position knowing distant to three known points with some error.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

To find position of a point we are looking for we can compute positions of points *I*, *II* and *III*. These points can be found by analyzing distances between unknown node and two known nodes (fig.5). For example coordinates of point $III(x_{III},y_{III})$ can be computed as follows:

$$x_{III} = x_1 + r_1 \cos(\alpha \pm \beta), \quad (5)$$

$$y_{III} = y_1 + r_1 \sin(\alpha \pm \beta). \quad (6)$$

where

$$\alpha = \arctan\left(\frac{y_2 - y_1}{x_2 - x_1}\right), \quad (7)$$

$$\beta = \arccos\left(\frac{r_1^2 - r_2^2 + d_{12}^2}{2 r_1 d_{12}}\right), \quad (8)$$

$$d_{12} = \sqrt{(x_2 - x_1)^2 - (y_2 - y_1)^2}. \quad (9)$$

Equations (5-8) are taken from triangle geometry.

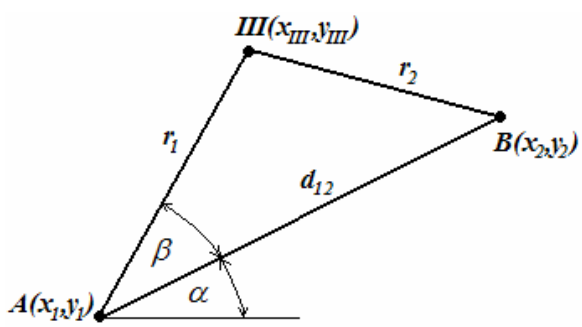


Fig.5. A method of finding unknown node position.

Two nodes with known positions (x_1,y_1), (x_2,y_2) and unknown node $III(x_{III},y_{III})$ form triangle in which all sides are known. Using cosines theorem we can write:

$$r_2^2 = r_1^2 + d_{12}^2 - 2 r_1 d_{12} \cos\beta, \quad (10)$$

which leads to equation (8). Equations (5-7) base on trigonometric relationships.

Knowing positions of points I, II and III we can estimate searched coordinates *(x,y)* as follows:

$$x = \frac{x_I + x_{II} + x_{III}}{3}, \quad (11)$$

$$y = \frac{y_I + y_{II} + y_{III}}{3}. \quad (12)$$

As we shown node III localization can be found using three transceivers with known positions and distances between them and III. Typical approach to localization in wireless networks is to put several number of base stations BS. BSs are often called "anchors" since their positions are known. One should deployed BS in such way that every possible point in the area has to be covered by at least three base stations. In this way any point can be localized. This solution is known from the GSM system [19] where BSs are used to estimate mobile stations' positions. In wireless sensor networks the easiest localization system requires to put three base stations around the field. If they can communicate with every node in the network then each sensor position can be determined. Unfortunately this method is very impractical since it is not always possible to cover all the area with only three BSs. On the other hand using larger number of base stations increases overall costs. Moreover the area where WSN operates may not be accessed (i.e. in military applications) and thus placing additional BSs may be impossible. What we propose is to transfer base station function to every sensor. Because sensors can communicate with each other they can also estimate distances between each other. In this way it is possible to localize one sensor using other sensors with known position.

3. Sensors localization in WSN

As mentioned above, wireless sensor network is build of hundreds of a homogenous sensors randomly deployed in some area. They can communicate with each other and estimate distance between them using some localization algorithm (e.g. RSS, ToA). It is important to mention that we do not have a limited number of sensors which collect data about distance to other sensors. On the other hand each sensor can find distance to all other sensors in its range (neighbors).

When network consist of *N* sensors then any nodes constellation can be fully described as *N* by *N* matrix. Elements of this matrix d_{ij} equal to distance between neighbor nodes *i* and *j* ($i, j = 1 \ldots N$), -1 if nodes *i* and *j* are too far to communicate and 0 if $j=i$. To find node *i* position it is necessary to know at least three $d_{ij} > 0$ elements where $j = 1 \ldots N$ with $j \neq i$.

If we want to determine sensors localization then position of one sensor (say *i*) should be assumed at first e.g. (0,0). Afterwards we assume position of another sensor *j* as $(0,d_{ij})$ where $d_{ij} > 0$. Using distances d_{ik} and d_{jk} ($d_{ik} > 0$ and $d_{ik} > 0$) we can find two possible positions of sensor *k* and one of these two positions should be chosen. In this way we assumed positions of three sensors. this allows to determine position of every sensor that can communicate with sensors *i*, *j* and *k*. Each sensor which position we have found in this way can be used to find another sensors' positions. If each sensor has connection with at least three other sensors (which we assume is always true because of redundancy in number of nodes in

wireless sensor networks), positions of all sensors can be computed.

Of course we cannot forget about what we did at the beginning. We assumed positions of three sensors so what we have computed is a sensors' constellation. In some application knowing absolute position is not necessary. We can imagine sensors with movement ability where network aim is to cover the area uniformly. In such situation knowing only constellation (without knowing sensors localization) would be enough. But generally we would like to know sensors localization which can be found by shifting, rotating or reflecting the constellation. To find the absolute positions we can put a BS and find distance to any three sensors.

Method presented above has advantages. Using e.g. humidity sensors, nodes can estimate attenuation and RSS algorithm will give accurate results. It is not necessary to put BSs around area (one BS will be enough). Typically clustering is applied in WSN so the network is divided into clusters with one Cluster Head and some number of Regular Sensors in each cluster [4,5,20]. Presented algorithm enables Cluster Heads to calculate nodes constellation in their cluster. This knowledge can be useful during re-clustering which is indispensable in WSN with clustering.

4. Error propagation in sensor localization

Using sensors as the BS to find other sensor localization has several advantages mentioned above. On the other hand if sensor's position estimation is erroneous then it will propagate and influence successive calculations. To avoid such error propagation we should carefully choose sensors when they are used to localize other sensors.

As we mentioned in section 2 to find sensor i localization using distances to nodes A, B and C with known positions, we have to find position of points I, II and III (see fig.4). Sensor i position (x,y) could then be found as follows:

$$x = \frac{x_I + x_{II} + x_{III}}{3},$$

$$y = \frac{y_I + y_{II} + y_{III}}{3}.$$

To describe what we call accurately of position estimation, we propose to compute estimation accuracy parameter e as follows:

$$e = a + b + c, \tag{13}$$

where

$$a = \sqrt{(x - x_I)^2 + (y - y_I)^2}, \tag{14}$$

$$b = \sqrt{(x - x_{II})^2 + (y - y_{II})^2}, \tag{15}$$

$$c = \sqrt{(x - x_{III})^2 + (y - y_{III})^2}. \tag{16}$$

For positions that appear to be quite accurate e is close to zero. Large value of e does not mean that estimated coordinates are wrong but it says that there were large discrepancies in computing points I, II and III.

Now let us discuss how parameter e can be used to reduce error propagation in sensors localization. We assume that sensor i can communicate with sensors $j_1 \ldots j_p$ that localization is known. To find sensor i position we use distances between node i and three of $j_1 \ldots j_p$ nodes. To reduce error propagation in sensor localization we chose three sensors with the lowest e parameter. In this way only 'reliable' nodes will be used to find other sensors position.

Parameter e is proposed to become criteria in choosing three nodes to find another sensor position. But we cannot forget about exceptions to the rule that to find sensor localization knowing three nodes is enough. When we use nodes A, B and C to find position of sensor P these exceptions are:

- when at least two of nodes A, B and C has the same localization,
- when all three nodes A, B and C lies on one line.

In these two cases it is not possible to compute node P position knowing only nodes A, B and C localization. To avoid these situations we propose to chose three of $j_1 \ldots j_p$ nodes as follows:

- choose sensor A with the lowest e parameter e_A,
- choose sensor B with the lowest parameter $e_B \geq e_A$ that is at least ε away from node A,
- choose sensor C with the lowest $e_C \geq e_B$ p in such way that the smallest angle in triangle formed by sensors A, B and C is bigger than γ.

This algorithm will help to avoid mentioned exceptions. Careful selection of ε and γ increases probability that three chosen sensors allow to determine node's P position even when positions and distances are estimated with an error.

5. Simulations results

5.1. Simulations

To evaluate proposed solution we made simulations in Matlab. In our simulations it was assumed that 250 nodes are randomly deployed into 1000 x 1000 field. Every sensor can communicate with other sensors if distance between them is smaller than sensor range R. At the beginning we assumed that positions of three sensors placed in one corner of the field are known. Moreover we know distances d_{ij}^* estimated by nodes, such that:

$$d_{ij}^{*} = \begin{cases} d_{ij} \cdot \left(1 + A_{ij} \cdot k\right) & \text{when} \quad d_{ij} \le R \\ -1 & \text{when} \quad d_{ij} > R \end{cases} \qquad (17)$$

In equation (17) d_{ij} is real, unknown distance between nodes i and j. A is a random number in range (-1,1) that refers to different accuracy of distance estimation and k is a parameter that changes from 0.005 (which means distances are estimated very accurate) to 0.1 (5-10% accuracy in distances estimation is reported to be possible to achieve [6,11,13]). Localization of all sensors will be determined using distances d_{ij} and our proposed algorithm.

In each iteration we take an unknown node i with the highest number of non-zero elements d_{ij}, where j is sensor with known position. We chose three known nodes j_A, j_B, j_C that meet conditions (ε and γ). Than we can determine node i localization and it can be used in next iteration to find another unknown sensor position.

After the positions of all sensors are determined we compute average error as follows:

$$AR = \frac{\sum_{i=1}^{250} \sqrt{\left(x_i - x_i^*\right)^2 + \left(y_i - y_i^*\right)^2}}{250}, \qquad (18)$$

where (x_i, y_i) is a real sensor position and (x_i^*, y_i^*) is estimated localization.

Tab.1. shows average results of 50 simulations for four different parameters.

Table 1. Simulations results for different parameters.

Parameters:	R = 500 ε = 75 γ = 15º	R = 500 ε = 50 γ = 15º	R = 500 ε = 50 γ = 10º	R = 250 ε = 50 γ = 10º
k = 0.005	10.049	10.343	11.076	8.304
k = 0.01	19.286	20.838	23.899	18.471
k = 0.02	34.519	34.787	44.499	33.390
k = 0.03	56.697	65.426	64.286	53.397
k = 0.05	111.998	112.713	112.009	106.443

Tab.1. shows relation between average error in sensors localization and distance estimation error (parameter k) for different R, ε and γ parameters. These parameters are related to method of choosing three sensors used to find other sensor position. It can be seen that for the same k, parameters R, ε and γ affect average error definitely.

We compared our method of sensor localization in wireless sensor networks with typical approach where three base stations (BSs) are placed on the edge of a field and distances to every node are measured. We consider tow cases. First when BSs are placed close to themselves (fig.6a). The other when they are placed in three different corners of the field (fig.6b). Our method is represented by parameters R = 250, ε = 50 and γ = 10º. Results are showed on fig.7.

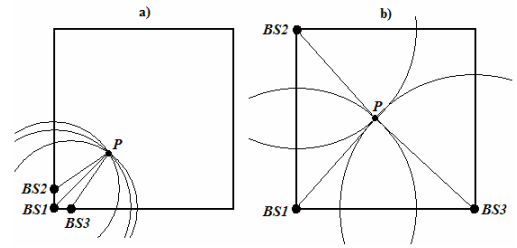


Fig.6. Base stations (BS) placed close to themselves (a) or in three corners of the field (b).

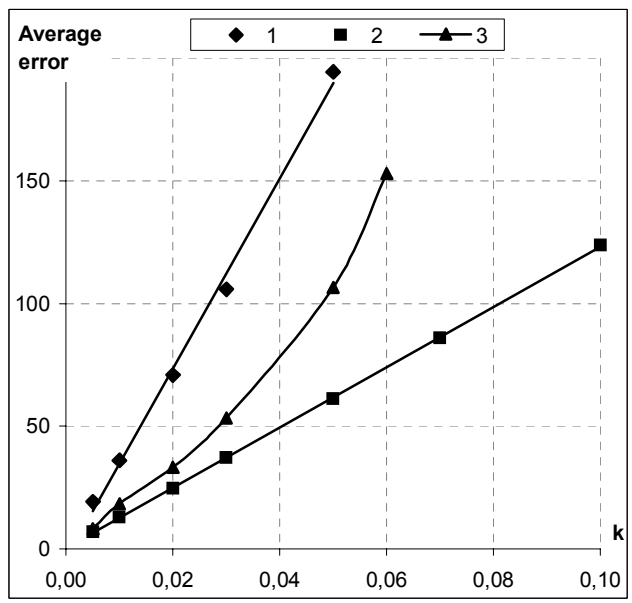


Fig.7. Simulation results. Sensors are localized using base stations placed in one of the corner (1) and in three different corner of the field (2) and using presented method (3) when R = 250, ε = 50 and γ = 10º.

Fig.7 shows, that when we use BSs placed in corners of the field to find sensors positions then we can significantly decrease position estimation error. It proves that one should avoid situations when nodes/BSs used to localize another sensor are close to themselves. We can also see that results achieved with presented method of sensor localization in wireless sensor networks are worse than those using base stations in three different corners of the field and much better than when base stations are placed close to one of the corners. As we see relationship

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

between average localization error (*AR*) and average distance estimation error (*k*) for methods that use base stations is linear. Because of error propagation in presented method *AR* grows faster when *k* increases.

5.2. Optimization technique

Efficiency of presented method depends on parameters *R*, ε and γ. Based on our simulation results we can conclude that:

- to minimize an error one should increase ε and γ. If three sensors are used to localize another one, error increases when these three sensors are close to themselves or they lie close to one line. Increasing ε and γ helps to avoid both situations.
- because we assumed that distance estimation error increases with distance, we should decrease *R*.

Of course if ε and γ increases to much then it my be impossible to find three sensors that meet up these conditions. The easiest way to solve this problem would be to use strict conditions first (relatively large ε and γ). If it is not possible to find three sensors that meet up these conditions we should decrease requirements until we found three sensors. Doing so we have modified algorithm of finding three sensors (that will be used to find node *i* localization) in following way:

1. put $z = 1$,
2. sort possible nodes $j_1 \ldots j_p$ (possible means those nodes that are closer than *R*) starting with sensor that have the lowest *e* parameter. In this way we have j_1 as a node with the lowest *e*, j_2 with second lowest *e*, etc.
3. choose $A = j_z$,
4. choose $B = j_m$ with $m>z$ that is at least ε away from node *A*,
5. choose $C = j_n$ with $n>m$ in such way that the smallest angle in triangle formed by sensors *A*, *B* and *C* is bigger than γ,
6. if three nodes were chosen we can compute position of other node *i* based on distances to nodes *A*, *B* and *C*,
7. if it was impossible to choose three sensors with $z < p$-2, increase *z* and repeat points 3-6,
8. if *z* reached *p*-2 and we do not have points *A*, *B* and *C* it means that we cannot find three nodes that meet up required conditions (ε and γ). Therefore we compute new values as follows:

$$\varepsilon = k_1 \cdot \varepsilon \,, \tag{19}$$

$$\gamma = k_2 \cdot \gamma \,, \tag{20}$$

where $k_1, k_2 < 1$ end repeat points 1-8.

Using presented algorithm we can find sensors *A*, *B* and *C*. By knowing their positions and distances to node *i* we can determine sensor *i* localization.

To evaluate proposed algorithm we compared simulation results with results achieved when three base stations were put in three different corners of the field and they were used to localize sensors deployed in the area (as on fig.7b). Parameters for our algorithm were: R = 250, $\gamma = 60^o$, $\varepsilon = R\sqrt{3}/2 \approx 288.68$, $k_1 = 0.95$, $k_2 = 0.975$. Parameter *R* was taken from previous simulations while ε and γ was set in such way that at the beginning, only sensors that form equilateral triangle inscribed in circle of radius *R* can be chosen to compute another node position. Several values of k_1 and k_2 were tried out and $k_1 = 0.95$, $k_2 = 0.975$ was found to give the best results.

We have also compared results achieved using AoA method. Sensors localization was found using two base stations placed in positions (-100,-100) and (1100,-100) as is shown on fig. 8. To localize nodes on a field we used α_1^* and α_2^*, such that:

$$\alpha_{1,2}^* = \alpha_{1,2} \pm k \cdot A \,, \tag{21}$$

where $\alpha_{1,2}$ are angles from unknown node to base stations 1 and 2 respectively, *A* is a random number in range (-1,1) and *k* is a parameter that changes from 0.1 degrease (very accurate angle estimation) to 10 degrease (values up to 10 are reported to be typical for AoA solutions [11,13,15]).

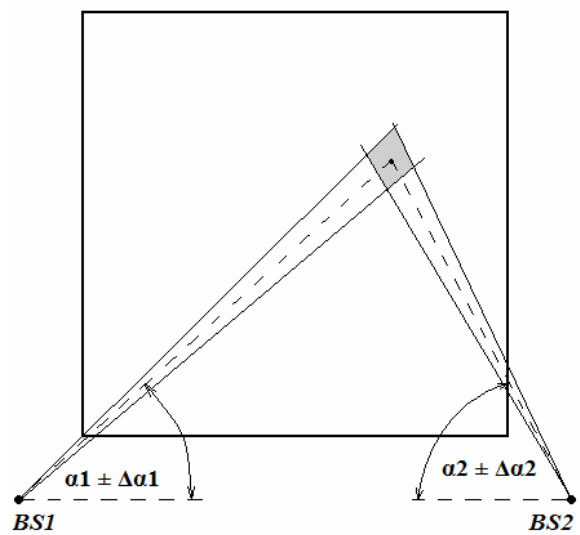


Fig.8. AoA method for sensors localization.

Fig.9 shows simulation results (average error in finding sensors localization in function of *k* or degrease). 3BS represents method where three base stations are placed in three different corners of the field. PM represents proposed method (R = 250, $\gamma = 60^o$, $\varepsilon = R\cdot\sqrt{3}/2 \approx 288.68$, $k_1 = 0.95$, $k_2 = 0.95$). AoA are results achieved using AoA algorithm as described above. As we see on fig.9 results achieved using proposed method with presented algorithm give smaller average error comparing to typical solution that uses three base stations or AoA method.

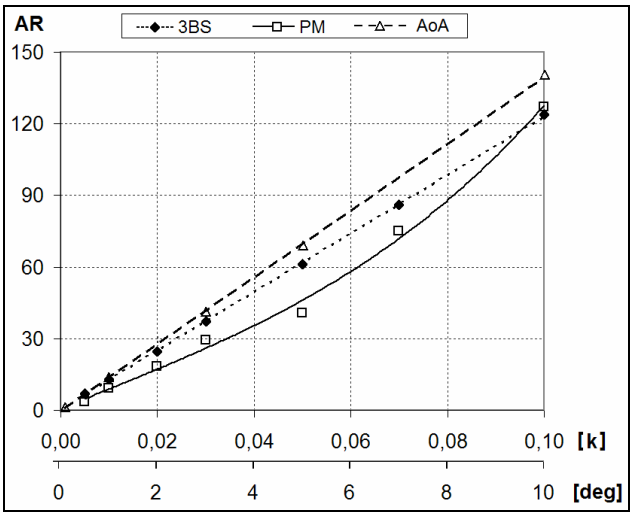


Fig.9. Simulation results. 3BS – localization using base stations placed in three different corners of the field, PM – localization using presented method (R = 250, γ = 60°, $\varepsilon = R\cdot\sqrt{3}/2 \bullet 288.68$, k_1 = 0.95, k_2 = 0.95), AoA – localization using AoA method.

As we present our proposal can easily compete with typical methods of finding sensors positions. It can be applied no matter of field size. It does not require to have fully access to the area what becomes very important e.g. in military applications. In some applications it is not necessary to use any base station because nodes constellation can by computed by sensors on their own. Average error in finding sensors positions is even smaller comparing to typically used methods.

As we can see the most critical part in presented method is an algorithm of choosing three sensors to find another sensor position and it will be a subject of our future work.

6. Conclusions and future work

In this paper we presented methods used to localize nodes in wireless networks. We described typical solutions that use base stations to find nodes localization. We proposed an algorithm dedicated to wireless sensor networks – structures build with hundreds and thousands of sensors that are randomly deployed in some area. In our solution base station function is transferred into every sensor. Nodes positions are determined based on distances to three other nodes with known positions (distances could be estimated using any possible method, e.g. RSS or ToA). We present simulation results that show how algorithm parameters affect position estimation errors. We optimized algorithm of choosing three sensors that are used to localize other sensor. Comparing our approach to solution that utilize BSs in the field corners and AoA algorithm our method is very competitive. It has several advantages and gives even smaller estimation error.

Presented method should be investigated using real-life sensors what will be a subject of our future work.

References

[1] M. Anlauff, A. Sunbul, *Deploying localization services in wireless sensor networks*, Distributed Computing Systems Workshops, 2004. Proceedings. 24th International Conference on

[2] A. Ault, X. Zhong, E. J. Coyle, *K-Nearest-Neighbor Analysis of Received Signal Strength Distance Estimation Across Environments*, 1st workshop on Wireless Network Measurements, 2005

[3] A. Cerpa, D.Estrin, *ASCENT: Adaptive Self-Configuring Sensor Networks Topologies*, IEEE Transactions On Mobile Computing, VOL. 3, NO. 3, Jul-Sep 2004

[4] Z. Chaczko, F. Ahmad, *Wireless Sensor Network Based System for Fire Endangered Areas*, ICITA 2005, Sydney, July 2005.

[5] Z. Chaczko, R. Klempous, J. Nikodem, M. Nikodem, J. Rozenblit, *An Improvement of Energy Aware Routing in Wireless Sensors Network*, European Modeling and Simulation Symposium, Barcelona, Oct 2006

[6] T.Y. Chen, C.C. Chiu, T.C. Tu, *Mixing and Combining with AOA and TOA for Enhanced Accuracy of Mobile Location*, IEE, Michael Faraday House, Stevenage, 2003

[7] L. Cong, W. Zhuang, *Hybrid TDOA/AOA Mobile User Location for Wideband CDMA Cellular Systems*, IEEE Transactions on Wireless Communications, Vol.1, No.3, July 2002

[8] P. Deng, P.Z. Fan, *An Efficient Position-Based Dynamic Location Algorithm*, IEEE 2000

[9] F. Gustafsson, F. Gunnarsson, *Positioning Using Time-Difference of Arrival Measurements*, Acoustics, Speech, and Signal Processing, 2003. Proceedings. (ICASSP '03). 2003 IEEE International Conference on

[10] M. Ilyas, I. Mahgoub, *Hanbook of Sensor Networks: Compact Wireless and Wireless Sensing Systems'*, CRC Press, 2005

[11] I. Jami, M. Ali, R.F. Ormondroyd, *Comparison of methods of locating and tracking cellular mobiles*, IEE, Savoy Place, London 1999

[12] D. Niculescu B. Nath, *Ad Hoc Positioning System (APS)*, IEEE 2001

[13] **N**. Patawari, A.O. Hero III, *Location Estimation Accuracy in Wireless Sensor Networks*, IEEE 2002

[14] N. Patwari, J.N. Ash, S. Kyperountas, A.O. Hero III, R.L. Moses, N.S. Correal, *Locating the Nodes. Cooperative localization in wireless sensor networks*, IEEE Signal Processing Magazine, July 2005

[15] R. Peng, M. L. Sichitiu, *Angle of Arrival Localization for Wireless Sensor Networks*, Third Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, (Reston, VA), Sep. 2006.

[16] I. Stojmenovic, *Handbook of Sensor Networks*, A John Wiley & Sons Inc., 2005

[17] R. Stoleru, J.A. Stankovic, *Probability Grid: A Location Estimation Scheme for Wireless Sensor Networks*, IEEE 2004

[18] A. Tarighat, N. Khajehnouri, A.H. Sayed, *Improved Wireless Location Accuracy Using Antenna Arrays and Interference Cancellation*, ICASP 2003

[19] A. Varshavsky, *Are GSM Phones the Solution for Localization?*, Mobile Computing Systems and Applications, 2006. WMCSA '06. Proceedings. 7th IEEE Workshop on

[20] O.Younis, S.Fahmy, *HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks*, IEEE Transactions On Mobile Computing, VOL. 3, NO. 4, Oct-Dec 2004

Adaptive misbehavior detection in Wireless Sensors Network based on local community agreement

Ryszard Klempous
Faculty of Electrical Engineering
Wroclaw University of Technology
Wrocław, Poland
ryszard.klempous@pwr.wroc.pl

Jan Nikodem
Faculty of Electrical Engineering
Wroclaw University of Technology
Wrocław, Poland
jan.nikodem@pwr.wroc.pl

Łukasz Radosz
Faculty of Electrical Engineering
Wroclaw University of Technology
Wrocław, Poland
lukasz.radosz@pwr.wroc.pl

Norbert Raus
Faculty of Electrical Engineering
Wroclaw University of Technology
Wrocław, Poland
norbert.raus@student.pwr.wroc.pl

Abstract

In this paper are discussed detecting misbehavior mechanisms in ad hoc wireless sensors' network. Term misbehavior is understood quite widely. A fake-positives, communication error, spreading false information are treated as sorts of misbehaviors. A cause of those phenomena is insignificant; it can be hardware failure, environment impact or purposeful hostile action. The main idea of this work is to propose efficient and simple algorithms capable to detect misbehaviors basing only on local community agreement. Additionally adaptive mechanism should provide isolation of faulty components. Using proposed mechanisms it is possible to reduce the disturbances, fake positives or invalid alarms in sensors fields.

1 Introduction

Over the past few years distributed wireless sensor networks (WSN) have been the focus of considerable research for both civil and military applications. Sensors are generally constrained in on-board energy supply therefore efficient management of the network is crucial to extend the life of the system. Failures and disturbance are inevitable in sensor networks executing in hostile environments and due to their unattended deployment [1]. To achieve greater scalability of WSN the main direction of technological progress in WSN filed is to self-organizing autonomous systems [12][11]. Such systems are vulnerable to misbehaviors of nodes. The main reason of such situation is lack of centralized management. To achieve greater robustness and stability of the system, it is necessary to add mechanisms which will work locally and will be able to detect and isolate hostile or faulty action. Those mechanisms will not only ensure local robustness but also increase whole systems efficiency. Faulty sensors or data distortion cause that whole network infrastructure may needlessly transfer erroneous or even totally false information through the entire system. Such events usually cost additional power consumption of key elements of WSN (e.g. Cluster Heads). Therefore stopping this misbehavior on local level is very important.

The main aim of this project is to explore and propose innovative computational and algorithmic solutions that can produce agreeable results in presence of uncertainties thus reducing influence of sensors that work incorrectly or sensors which capture fake events due to disturbance or other environmental factors. The general concept of explored algorithms is based on local community agreement [8][9] and adaptive mechanisms which are based on simplify artificial immune systems [10]. This work was also inspired by mechanisms used in Byzantine Fault Tolerance [6].

2 Proposed Solution

Mechanism is based on local community agreement; local community has the best knowledge about situations in neighborhood area. Only local community can force the system to ignore misbehavior or hostile action. In Wireless Sensors Network it is quite hard to differentiate between a proper action or

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

misbehavior, because both have local range. But nowadays designs of WSN fields assume that sensors' ranges are overlapping, and one event should be detected by few sensors. Assuming that fact it is possible to locally exchange information in order to decide if detected event in fact occurred or if it was a misbehavior caused by local interference or other factor.

Decision is based on measure defining reliability level of an event. The measure was called Event Reliability Factor (ERF*)*. It is *computed* by every sensor in local community area. To append adaptive mechanisms to agreement algorithm each node keeps a reliability level *NRF* (Node Reliability Factor) of surrounding sensors. After every agreement process those coefficients are modified and updated. Sensor can make a decision to ignore the node which reliability ratio is lower than some predefined value.

So operation of adaptive misbehavior algorithm can be divided into three simple steps.

Step one: *(Detection of event)*. An event is detected by some sensors. It is not known if the noticed event is true incident or misbehavior. The adaptive misbehavior algorithm is being initialized.

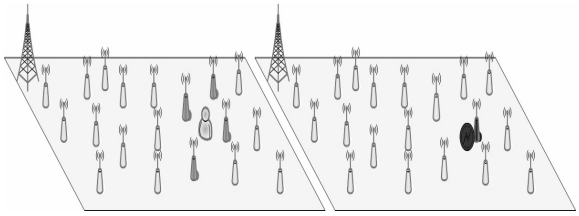


Figure 1. Local event detection.

Step two: (*Local communication)*. An exchange of information takes place within the surroundings of an event. Nodes collect information from each other and compute event reliability factor ERF.

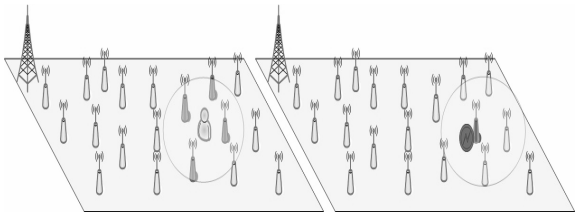


Figure 2. Local communication

Step three: (*Local decision and action*). Based on previously computed event reliability factor (ERF) each node classifies an event as a true event or misbehavior. True events are send further to the network (e.g. to Cluster Heads) while misbehaviors are blocked at the local level. If one of the nodes continuously tries to send some false information, its reliability level is decreased, and after reaching a frontier value it can be permanently excluded from communication (alerts raised by this sensor are ignored).

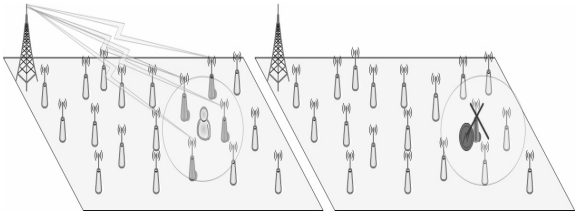


Figure 3. Local decision

3 Mechanism's Description

3.1 Used symbols and abbreviations

ERF - Event Reliability Factor

M(i) – message send by the i-th sensors

M(i) = <EventID, NodeID, Info>

M(i).EventID – Id of the event, each event has its own ID

M(i).NodeID – Id of the node which is broadcasting the message

M(i).Info – Information send in the message.

M(i).Info ∈ {0,1}

M(0) – information detected by current sensor

NRF– Node Reliability Factor (*NRF*)

AR – Agreement Range (Radio communication range)

SR – Node Detection Range (Sensor's Range)

LAT – Local Agreement Table – used to collect information received from surrounding sensors.

LNRT – Local Nodes Reliability Table

ECT – Event Classification Threshold

3.2 Basic assumption:

- Nodes should be able to communicate with others surroundings nodes in a flexible manner (broadcast communication)
- Nodes should be capable of receiving and sending broadcast messages.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

- Nodes should be able to change communication signal strength in order to configure communication range.
- Every part of field should be covered by detection range of several nodes.

3.3 Nodes' state dependency chain

During execution if proposed algorithm following nodes' states can be distinguished.

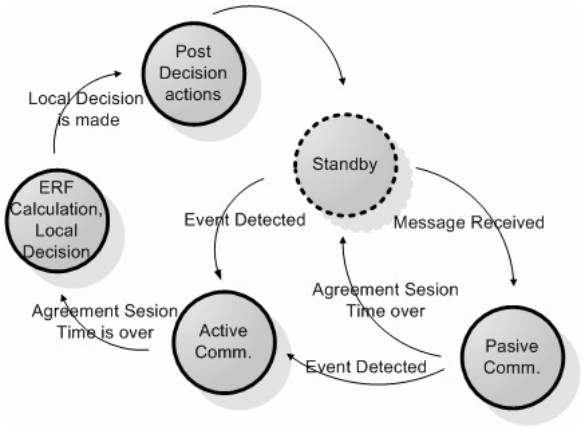


Figure 4. Nodes' state dependency chain

Standby state – in this state all nodes are waiting for activation. Node can be activated in two ways: the event can by detected or message can be received.

Active Communication State – in this step the whole local communication is proceeding. To enter this state, the event must be detected or the message about detection of the event must be received. It is possible to enter Active Communication State directly from the Standby State or from the Passive Communication State.

Passive Communication State – node enters this state when it received a message about not detecting the event. This state is typical for nodes on the border of local community. It is possible for node to stay in this state to the end of the Agreement Session and return to standby state or to cross to the Active Communication State.

ERF Calculation and Local Decision – node enters this state after successful information exchange (Active Communication State). In this state each node calculates ERF and makes a decision about occurred event.

Post Decision Actions State – node enters this state after having made up the decision about occurred event. Action carried in this state depends on the decision. It could be:

- Sending the information about event further in the network
- Nodes Reliability Factors Update
- Feedback mechanism

3.4 Adaptive misbehavior detection mechanism

3.4.1 Building local community

Local nodes' community is build using messages broadcasting within defined communication range AR. It is assumed that communication range should be similar to detection range of the sensors $AR \sim = SR$. To obtain desired communication range a communication signal strength adjustment can be utilized.

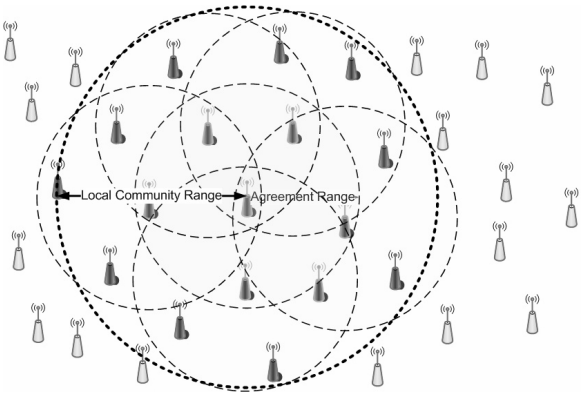


Figure 5. Local Community visualization

Building optimal Local Community is crucial for efficient work of the misbehavior detection mechanism.

3.4.2 Local Information Exchange (Agreement Session)

During Agreement Session, nodes exchange information about the occurred event. Two types of node's state during Local Information Exchange can be distinguished.

1. Receiving and transmitting state (active state) – node is in this state when event was detected by the current node or when the node has received the message about the occurrence of the event.

2. Receiving state (passive state) – node is in this state when it receives a message informing that the event has not occurred.

When node is in active state, it is clear that it should broadcast information about the detection of event regardless of the payload of the message. In passive state the messages are only received, non information is send unless node's state is changed (state can be changed in case of detection the event or receiving the message with information about detecting the event). In both states all received messages are memorized. All received data are kept in Local Agreement Table by each node. In case when Agreement Session is over and the node state is still passive, whole received information is deleted and node returns to the standby state. When node is in active state after successful agreement session the Event Reliability Factor calculation step is initialized. There is problem with distinguishing one agreement session form another. In this work is proposed to use time to differentiate one Event from another. It is supposed that all received information about the events should be in some defined period of time and that are treated as the information about the same event. It is also proposed to identify each event with unique ID. Node which has first detected and broadcasted an information about the even to local community, defines the ID; other nodes intercept this ID and mark an event using this defined ID in further communication.

3.4.3 Event Reliability Factor calculation

After successful information exchange each node having taken part in Agreement Session acquires a LAT table with information about occurred event, to be precise each node has knowledge about state of surroundings sensors. Table below shows exemplary contents of LAT table.

Node ID (i)	Event ID	Info
342	34	0
234	34	0
644	34	0
341	34	1
235	34	0

TABLE I. LOCAL AGREEMENT TABLE

Reliability factors are computed utilizing this LAT table. In calculation process it is also used Local Node Reliability Table (LNRT). The LNRT contains the reliability factors of surrounding nodes; those factors can be change in order to adapt the algorithm to the new situation. In the beginning stage all NRF's are initialized with 1. During adaptive mechanism those factors are changing. The following expression is used to compute Event Reliability Factor (ERF):

$$ERF \in <0,1>$$

$$NRF \in <0,1>$$

$$Info \in \{0,1\}$$

$$ERF = \frac{\sum_{i=1}^{n} M(i).Info \cdot NRF_i}{\sum_{i=1}^{n} NRF_i}$$

M(i).Info – Information received from the i-th node

NRF_i – Reliability factor of the i-th node. Future decision about event classification is based on this factor.

Using data from LAT shown above and data from LNRT shown below the ERF amounts 0.2.

Node ID (i)	NRF(i)
342	0.9
234	0.9
644	0.8
341	0.9
235	1

TABLE II. LOCAL NODE RELIABILITY TABLE

3.4.4 Local Decision

After Event Reliability Factor calculation the decision is made. Classification of the event is simple. Event is recognized as true if Event Reliability Factor is equal or greater than some defined value. This value is called Event Classification Threshold ECT. It is assumed that this value should be set to 0.5. If it was decide that event is true, Nodes which have detected this event are allowed to send an information about this event further in the wireless network (e.g. to Cluster Head). But if event was recognized as false, it means that network deals with misbehavior of the node and further communication should be blocked.

3.4.5 Nodes Reliability Factors Update

After phase of Local Decision, each node updates its Local Node Reliability Table (LNRT). If the event

was recognized as false, reliability factors of nodes which sent different message are decreased. In case of true event detection, reliability factors of nodes in Agreement Session which sent information about detection of event are increased. This reliability factors are kept locally by the each node. Currently there is no Nodes Reliability Factors exchange mechanism proposed.

If $ERF >= ECT$
  For each i in LAT
    If $M(i).Info == 0$ then $NRF_i -= \Delta NRF$
Else
  For each i in LAT
    If $M(i).Info == 1$ then $NRF_i += \Delta NRF$
End If

Additionally a notification mechanism is proposed, if one of the node's reliability factor is below some defined level, nodes which have detected such situation can notify e.g. Cluster Head about such situation. Such node can be excluded from the field or some repair mechanisms can be applied (e.g. the reboot command can be sent to such node).

3.4.6 Feedback mechanism

Some feedback mechanism is necessary to obtain greater stability of local sensors community. Because in this during Nodes Reliability Factors Update is quite often that sensors on borders of Local Community incorrectly update the nodes reliability factors. Such situation can be corrected easily. It is only necessary that each node is able to listen and try to detect a message send after successful Local Decision. This message is supposed to be received by cluster head, but there is no problem to detect this massage by the nodes. When the node detect the final information about the detected event, it is possible to note down this fact and use this information to correct mistakenly updated nodes reliability factors.

3.1. Algorithm Diagram

Below is shown a proposition of algorithm that should be applied for every node to obtain functionality proposed in this paper. After initialization node's action can be triggered in both ways, node can collect information from sensor about detection of an event or it can receive a message from other node. Whole communication is based on broadcasting. Using Radio broadcast with defined signal strength is possible to build local community. Using the Timer and ActiveID identifier it is possible to distinguish one agreement session form another. ActiveID field indicates the active event's agreement session. Whole local information exchange is simply based on receiving information from other nodes, adding it to local agreement table and broadcasting own information. After information exchange each node computes Event Reliability Factor, takes local decision and then an action is executed (sending or not sending the message further in the wireless network). After this it is possible to update a local nodes reliability table and if necessary correct it.

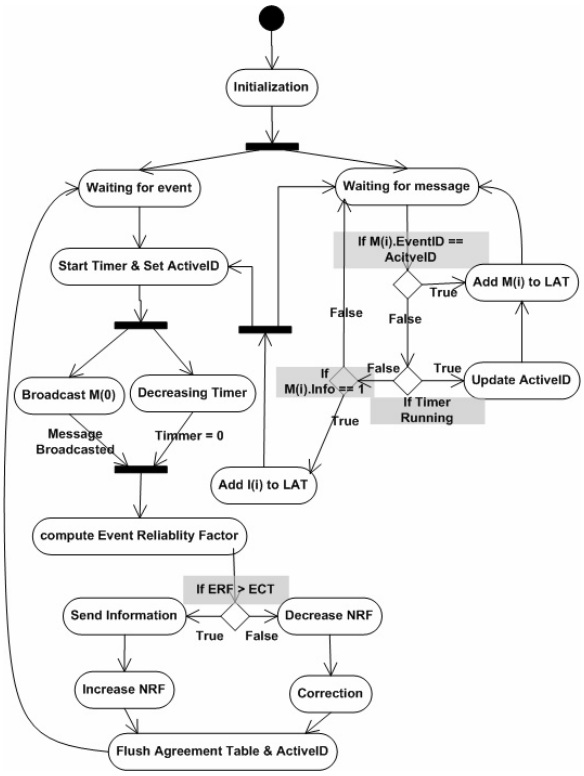


Figure 6. Algorithm's Diagram

4 Simulation

4.1 About Simulation

With the object of initial testing proposed mechanism, the simulator was build using Matlab environment. During numerous performed simulations an optimal set of arguments was established. It was discovered that an Agreement Range (AR) should be set to about 80% of Sensors Range. Using this setting a number of false positives are minimized. Sensors Range should be set to guarantee covering of every

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

field's part with sensing range of several sensors. That means the proper sensors density should be assure. In simulation it was used 1000 nodes spread on field of 1000000 units², Sensing Range radius was set to 50 units and Agreement Range radius to 40 units.

4.2 Simulation's Examples

Below is shown some simulation examples. On the three dimensional plots the distribution of event reliability factor is being shown. The factor distribution is shown with reference to the event classification threshold and in absolute way. On figure showing sensor field, marked dots are the nodes which took place in Local Agreement. The x shows the location of the event. The dots surrounded by circle are nodes which send information about detection of the event.

4.2.1 True event correctly classified

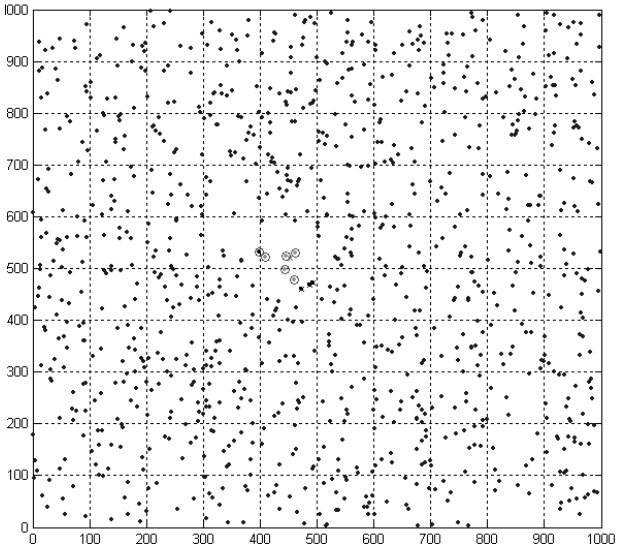


Figure 7. Sensor field with simulation of the event

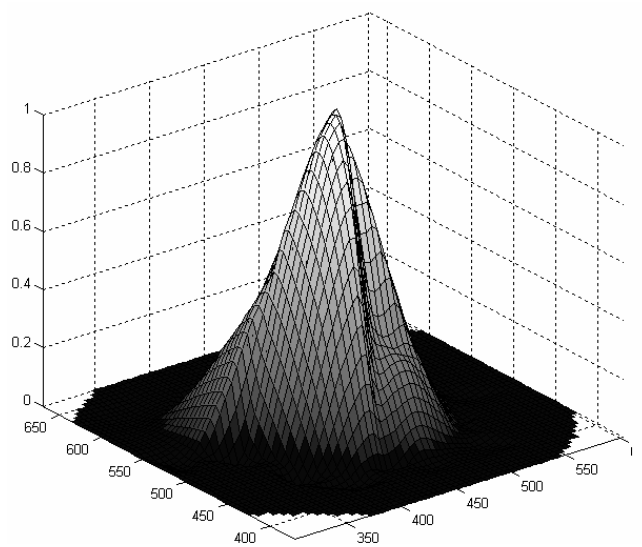


Figure 8. Absolute distribution of event reliability factors

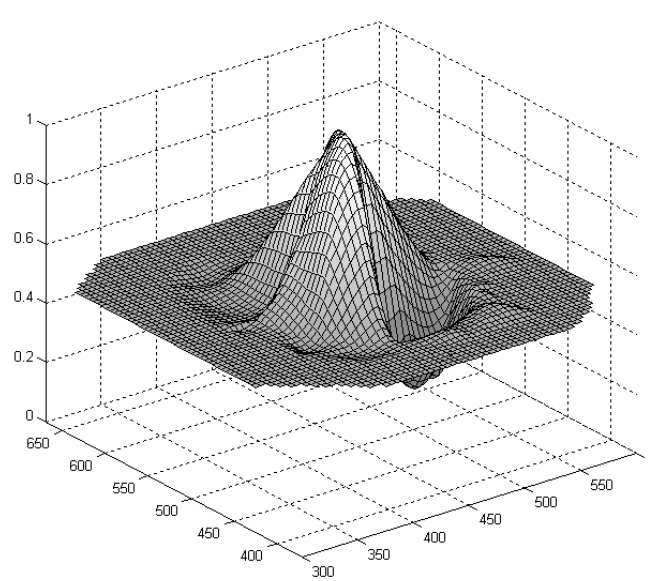


Figure 9. Reliability factor distribution with reference to the event classification threshold

On the figures above it is shown the distribution of the events' reliability factors. As it is easily to observe that the reliability factor is above the Event Classification Threshold (*ECT*=0.5). The location of the maximum of the factor shows us the approximate position of the event.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

4.2.2 False event correctly classified as misbehavior of the node

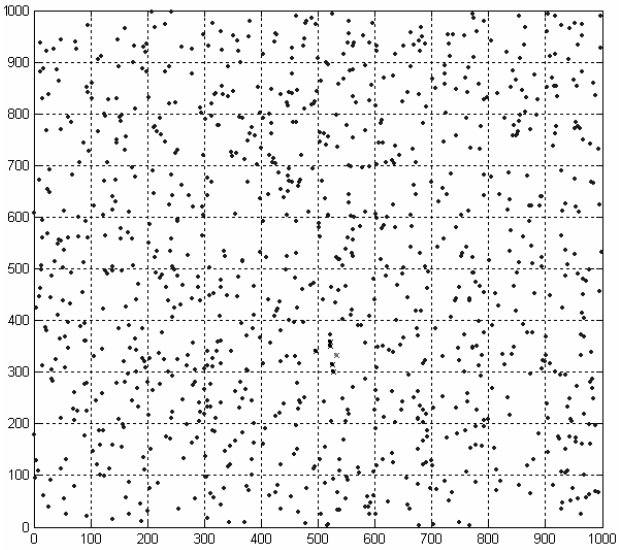


Figure 10. Simulation of misbehavior

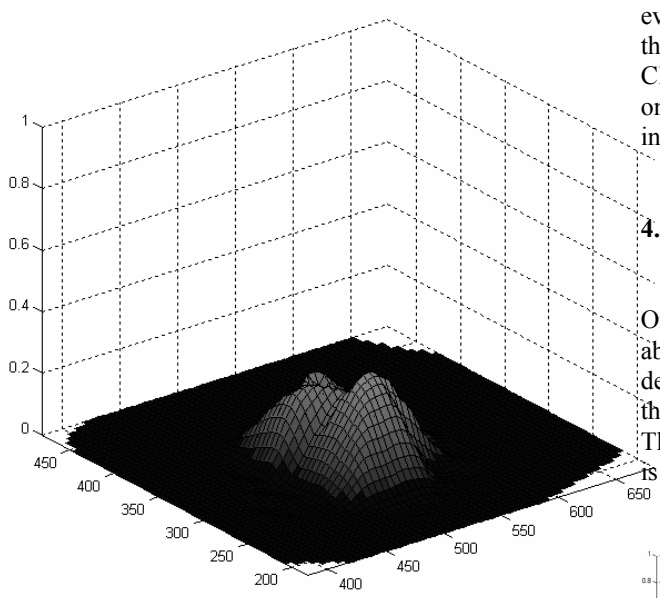


Figure 11. Reliability factor distribution with reference to the event classification threshold

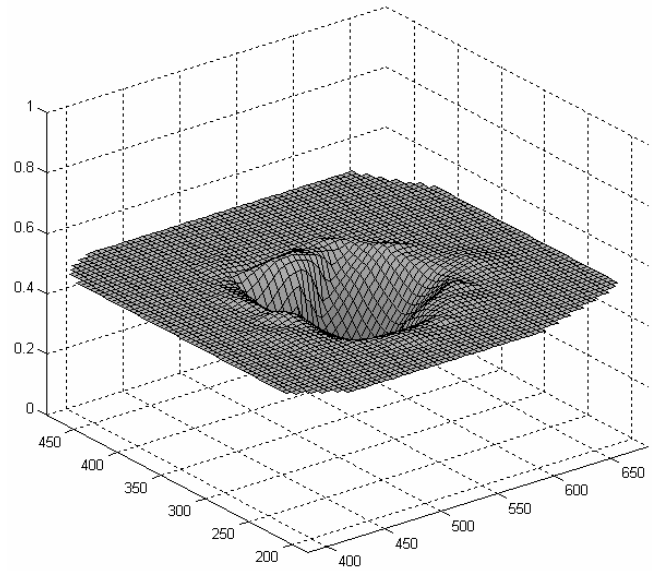


Figure 12. Reliability factor distribution with reference to the event classification threshold

On the figures above it is shown the distribution of events the reliability factors. As it is easily to observe that the reliability factor is far below the Event Classification Threshold (ECT=0.5).This means that one of the node misbehaves and trays to send false information

4.2.3 Adaptation process

On figures below is shown an adaptation process. One of the sensors continuously sends the information about detection of the event, surroundings sensors decide that no event has occurred. In consecutive steps the reliability factor of misbehaving node is decreased. Thus whole distribution of the event reliability factors is also decreased.

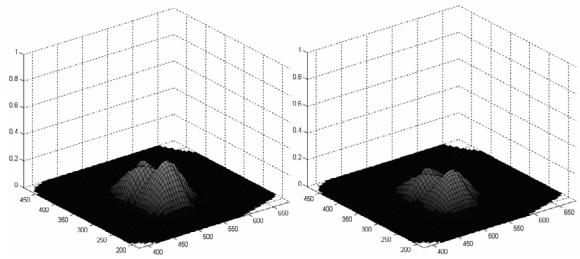


Figure 13. Adaptation process; step one and two

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

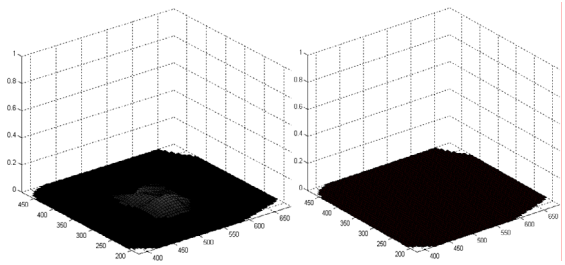


Figure 14. Adaptation process, step three and four

At the last figure we see that the signal broadcasted by the misbehaving node was completely ignored.

5 Conclusion

The main idea of this work was to extend Wireless Sensors functionality by adding some misbehavior detection mechanisms. The key task of these mechanisms is to reduce influence of single disturbance or false alarms in the field of sensors. The core assumption was to do it locally in the sensor field, because WSNs becoming more and more autonomous so the mechanism used in such networks should be independent of centralized management. The majority of decision should be performed on a local field. This idea can be compared to a society that locally copes with problems using basics of democracy. To perform this task we used an algorithm which development was inspired by solutions known form false tolerance commonly named the Byzantine False Tolerance, Artificial Immune Systems and Swarm Intelligence. Despite of a rather high cost of additional communication we attempted to add those properties to Wireless Sensors Networks. Our innovative approaches and positive results of our initial simulation gives us a certain level of confidence that we are on a right track however further real-life experiments are required. In our future work we are thinking about adding some additional properties for our algorithmic solutions. For example, by adding some pattern recognitions know from the adaptive Artificial Immune Systems.

6 References

[1] Z. Chaczko,F. Ahmad, "Wireless Sensor Network Based System for Fire Endangered Areas" , ICITA 2005, Sydney, July 2005.

[2] Z. Chaczko, S. Sinha, "SAFE Model Approach to Construction of Intelligent Security Systems", ICITA 2005, Sydney, July 2005.

[3] S.C. Wang K.Q. Yan H.C. Hsieh, "The Byzantine Agreement under Mobile Network", International Conference on Networking, Sensing & Control ,Taipei. Taiwan. March 2004

[4] B. Liskov, R. Rodrigues, "Tolerating Byzantine Faulty Clients in a Quorum System", ICDCS'06, Lisbon, 2006

[5] V. Drabkin, R. Friedman, Marc Segal, "Efficient Byzantine Broadcast in Wireless Ad-Hoc Networks", DSN'05, Israel, 2005

[6] L. Lamport, R. Shostak, M. Pease, "The Byzantine Generals Problem." ACM Transactions on Programming Languages and Systems, 1982.

[7] C. Liskov, "Practical Byzantine Fault Tolerance". In *OSDI: Symposium on Operating Systems Design and Implementation* (1999), USENIX Association, Co-sponsored by IEEE TCOS and ACM SIGOPS.

[8] M. Pease, R Shostak, and L. Lamport,"Reaching Agreement in Presence of Faults," JACM, vol. 27, no. 2, pp. 228234, April 1980.

[9] J. Martin, L. Alvisi, M. Dahlin, "Small Byzantine Quorum Systems", International Conference on Dependable Systems and Networks, DSN'02, 2002

[10] P. SU, D. Feng, "The Design of an Artificial Immune System", International Conference on Networking, International Conference on Systems and International Conference on Mobile Communications and Learning Technologies, 2006

[11] F. Silva, T. Regina, M. Braga, L. B. Ruiz,J. Marcos S. Nogueira, A. Loureiro, "A Comparative Study of Distributed Self-management Approaches for Wireless Sensor Networks", 2006

[12] Falko Dressler, "Benefits of Bio-inspired Technologies for Networked Embedded Systems: An Overview",

Design of A Wireless Sensor Network Based Automatic Light Controller in Theater Arts

Chuan Feng, Lizhi Yang, Jerzy W. Rozenblit
Electrical and Computer Engineering Department
The University of Arizona
Tucson, Arizona 85721
Email: {fengc, lyang, jr}@ece.arizona.edu

Peter Beudert
School of Theatre Arts
The University of Arizona
Tucson, Arizona 85721
Email: pbeudert@email.arizona.edu

Abstract

In this paper, an automatic lighting controller designed and built using a wireless sensor network indoor positioning technology is described. This controller can autonomously track actors during a real-time theatrical performance. Kalman filter and 3-D trilateration technologies were used with Cricket wireless sensors to implement this system. In addition, an entertainment industry standard protocol DMX-512 and an efficient calibration method were applied to realize remote computerized fixture control. As far as we know, this controller is the first application of wireless sensor networks in the theater arts area. A successful public performance concert at the University of Arizona validated the performance of the system.

1 Introduction

In this paper, we propose an automatic moving light controller which was built using a wireless sensor network (WSN for short) based indoor positioning technology. WSN is an interconnected wireless system of typically small low power electronic sensors that provide ubiquitous sensing and computing capabilities, through which the controller tracks the position of actors on the stage. It controls the light instruments to illuminate the actors as needed [2]. In contrast with traditional systems that rely on preprogrammed sequences, this system provides autonomous abilities to control the moving lights in real-time.

Modern stage lighting plays an important part in a performance. The director can use the lights to alter the perception of shapes, direct audiences' attention, set the mood of a scene, establish position in time and day, and trigger a variety of events.

To achieve these various objectives, different types of lighting instruments and related control systems are implemented. One modern instruments which appeared in theaters in the early 1980's is an intelligent fixture (or moving light) . An intelligent fixture allows remote control of the movement of a light beam either by moving a mirror that reflects the beam in front of the lens, or by moving the whole fixture itself.

Traditionally, the moving light is pre-programmed by the lighting designer before a real time performance [9]. It is like an industrial manipulator that can only play-back what has been memorized during a training session. There are no automatic lighting devices that can track and follow actors during a show because they lack real time position information. We wanted to build a system which obtains the position of actors in real time and controls the beam of the intelligent fixture automatically.

In the next section, we briefly review previous work. In Section 3 we describe the High Accurate Positioning subsystem including the wireless sensor network, Kalman filter, and the 3-D trilateration algorithm. In Section 4, we present the automatic lighting control subsystem. Section 5 describes the whole system setup and implementation. In Section 6, the experiment and real performance results in the theater of the University of Arizona are presented. Section 7 discusses future work.

2 Related Work

Theater lighting is one of the most important features in modern theater performance. In Disney's "Beauty and the Beast", which opened at the Dominion Theater in London 1998, there were 1240 lighting units, with different levels of intensity, changing color and focus on moving objects at different times [6]. In that performance, sixty seven automated fixtures were used, which communicated with the control desk via the DMX 512 protocol, an entertainment industry standard based on RS485 to change the pan, tilt, intensity, color and gobo. These fixtures usually utilize com-

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

pact arc lamps as light sources. They use servo motors or stepper motors connected to mechanical and optical devices to manipulate light before it emerges from the fixture's front lens. Currently, there are several different manufacturers working in the market, such as Highend, Vari Lites, SGM Light Technology etc. In our project, a Highend Studio Beam system was used.

Besides the lighting system, another necessary component of the automatic lighting device is the positioning system. Today, thanks to satellite navigation systems such as GPS, the outdoor positioning is reliable and convenient [10]. However, although GPS signals are free to everyone, there are blind spots, especially inside buildings, and the accuracy of a GPS tracking system is inadequate for indoor applications.

There are several indoor positioning technologies available [3] [4] [5] that can offer higher accuracy than the outdoor location systems within a confined space. One system uses an RF signal of wireless AP or Bluetooth devices to locate objects. They measure distance between the transmitter and the receiver based on Received Signal Strength Information (RSSI). Another measurement technology is called Time Difference of Arrival (TDOA), which can be implemented by ultra wide band (UWB) technology or ultrasonic pulse. Image processing is another method. Among all the available techniques, a wireless sensor network is a relatively new and advanced technology because the wireless sensor networks integrate technologies from sensing, communications and computing to small individual sensors which can be deployed into the indoor environment flexibly.

The location accuracy requirements of an indoor positioning system can be grouped into three classes: low accuracy at room-scale, medium accuracy around 1 meter and high centimeter-scale accuracy. The requirements of the automatic lighting system are reliable coverage and high accuracy within 10 centimeters. RSSI technology does not require complex hardware, but it is difficult to measure precise distance and estimated position. Image processing technology cannot adapt to complex lighting and background variations in a theater environment. Ultrasonic pulse TDOA devices offer high accuracy at a few centimeters without requiring complicated hardware such as UWB. Thus, they are a suitable positioning part of the lighting system. The Active Bat and Cricket systems are two successful products of this type. For this project, we used the Cricket system.

The Cricket Location-Support System was first developed at MIT's Artificial Intelligence Laboratory and is now distributed by Crossbow Technology. The Cricket system is a decentralized wireless sensor network which tracks objects using the TDOA between radio frequency and ultrasonic signals, which provides high accuracy distance measurement (within 1 cm). By using MICA2 platform [11], we can build a small wireless sensor network that consists of several fixed mounted beacons, and one listener which can be carried by the actor to achieve tracking goals. Detailed information about the Cricket application will be provided later in this paper.

3 High Accuracy Positioning

Because theater applications have high tracking accuracy requirements, Cricket is used as the positioning sensors in the automatic lighting project. Cricket consists of nodes which are small wireless sensors. Each node consists of a radio frequency (RF) transceiver, an ultrasonic sender and receiver, and a microcontroller. The positioning system uses distance information acquired from Cricket nodes to estimate the exact 3-D coordinates of the actor.

3.1 Wireless sensor network

There are two kinds of Cricket nodes: 1) beacons that usually act as reference points of the location system and are typically attached to the ceiling of a building, and 2) listeners that are attached to actors so that the system can determine their location. Beacons periodically transmit RF messages containing the unique beacon-identifier (ID). At the same time, they also send an ultrasonic pulse. The transmission is not centrally coordinated. Listeners listen to beacon transmissions and measure distances to nearby beacons by comparing the time difference of the RF message and the ultrasonic pulse. Then the listener transfers the estimated distances to a super node through a bluetooth adaptor. The super node is a computer that has computational power for further operations including trilateration and lighting control. Fig. 1 is the schematic diagram of the wireless sensor network.

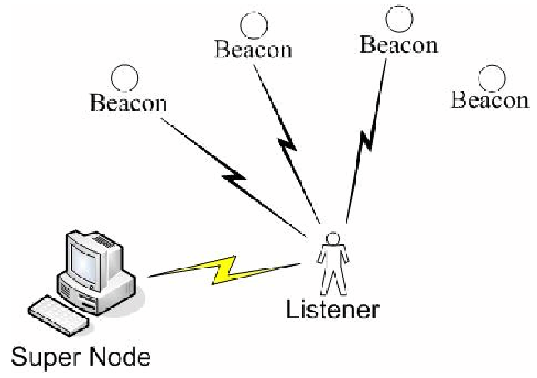


Figure 1. Cricket Sensor Network

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

3.2 Kalman filter

The Kalman filter is one of the most popular mathematical tools used for noisy sensor measurement by stochastic estimation [11]. In practice, it is difficult to get the an exact solution because of noise and device errors. The noise of the distance time difference of ultrasonic and RF signals mainly comes from sound reflection or other effects. The Kalman filter is an efficient recursive filter that estimates system states from sequential noisy measurements by exploiting the dynamic of the target. The system maintains a table of the distance estimations between a listener and beacons in realtime. When a listener receives distance information from a beacon, the system uses the error between the current distance information and the new one to update the distance table. A Kalman filter is implemented as the correction phase.

Following are the system time update equations.

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_k \tag{1}$$

$$P_k^- = AP_{k-1}A^T + Q \tag{2}$$

$$z_k = Hx_k + v_k \tag{3}$$

Where x_k is the system state, $\hat{x}_k^-$ is a priori state estimate at step k, and $\hat{x}_k$ is the posteriori state estimate at step k. P_k^- is a priori estimate error covariance and P_k is the posteriori estimate error covariance. A is the state equation that describes the relation of the previous state and current estimated state. B relates the previous distance difference u to the state x. In this application, the matrix A and B are I; Q is the process noise covariance; z_k is the measurement value, v_k is the measurement noise and H is the measurement matrix, which is also equal to I in this application. In practice, z_k can be measured directly, so equation (3) is not necessary, but the measurement innovation $z_k - H\hat{x}_k^-$ is important in the measurement update step.

The Kalman gain equation is shown in the following:

$$K_k = P_k^- H^T (HP_k^- H^T + R)^{-1} \tag{4}$$

Where R is the measurement noise covariance.

According to the time update equations and the Kalman gain K_k, the estimate state and error covariance can be updated by equations (5) and (6).

$$\hat{x}_k = \hat{x}_k^- + K_k(z_k - H\hat{x}_k^-) \tag{5}$$

$$P_k = (I - K_k H)P_k^- \tag{6}$$

Fig. 2 is the graph of a simulation of distance measurement. The x-axis is simulation time and the y-axis is distance information. The "+" indicates information got from the sensors, the curve shows the estimated distance. The Kalman filter can indicate distance without being influenced by noise.

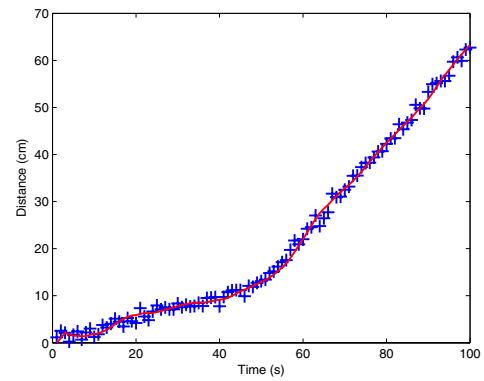


Figure 2. Kalman Filter Diagram

If error is significant, there will be a big shock wave as shown in Fig. 3. In that simulation, at t=10s, an error measurement of around 100 centimeters was received. Although the system steadied after a while, the big shock is not acceptable.

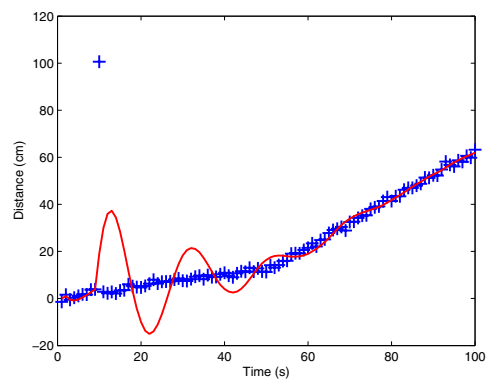


Figure 3. Error Data Effect to Kalman Filter

To minimize such effects, a logical estimation is applied in addition to the complete computation iteration of the Kalman filter. The distance from the sensor will be considered as incorrect and removed if it is too far away from the estimated distance. However, from [1], a serial small distance error may cause the Kalman filter to go into a bad state that treats all of the correct distance information as incorrect. So if continuous incorrect distance information is detected, the filter will know it is in the bad state because the integration of the error will be greater than a predefined threshold. Therefore, the system will clear all the information and start the filter from the initial state again. Using the self-reset mechanism, the filter can eliminate the serious infection from incorrect data.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

3.3 3-D trilateration

The object of the positioning system is to acquire the coordinates of the target. When using the Kalman filter, the distances of the target (listener) and reference points (beacons) can be known. Trilateration is a method to determine the object position by some reference points with known locations. Like a GPS, the 3-D space tracking system needs trilateration to detect the position of an object. Assume the object is at position P_0, there are known reference points P_1 to P_n, and all of these points are in 3-D Cartesian coordinate system. If the precise distances from the reference points to the object are known, equations can be derived as follows:

$$\begin{cases} |P_0 - P_1| & = & D_1 \\ |P_0 - P_2| & = & D_2 \\ & \dots & \\ |P_0 - P_n| & = & D_n \end{cases} \tag{7}$$

Where $P_0 \dots P_n$ are the coordinate vectors of the object and reference points and $D_1 \dots D_n$ are the distances between the object and reference points. In a general 3-D application, 4 reference points are needed to solve the equations. If the redundant solution can be eliminated easily, 3 reference points are enough, e.g., mounting the sensors on the roof and tracking people walking on the floor, two different coordinates will be observed by the trilateration of the 3 reference points, but one of them will be removed because it is over the roof.

Theoretically, the coordinate information of the target can be obtained by solving the equations 7. But device errors or inaccuracy cannot be eliminated by the Kalman filter. There is an example in Fig. 4.

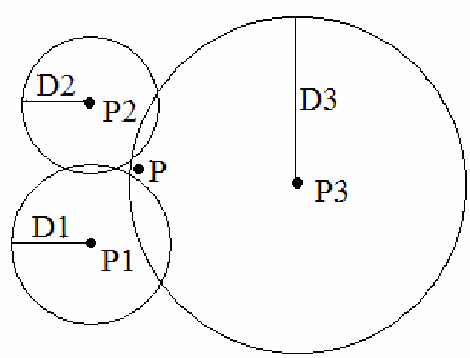


Figure 4. Trilateration Problem

There are many algorithms that have been proposed for solving nonlinear equations. The least square algorithm is a popular one that can be adopted to acquire the approximated solution of trilateration. In order to estimate the 3-D position from three measured distances D_i, the algorithm updates the position estimate $\hat{P}$ to minimize the cost function E. The cost function is the sum of the errors between the measured and the estimated distance.

$$E(\hat{x}) := \sum_{n=1}^{3} (D_i - |P_{\hat{x}} - P_i|)^2 \tag{8}$$

In equation (8), D_i is the distance information obtained from the sensor, $P_{\hat{x}}$ is the estimated position of the target and P_i is the position of the reference points. In order to solve the nonlinear problem, the Newton method is applied.

The Newton method is an efficient root-finding algorithm for solving nonlinear equations. The idea is as follows: one point is selected from the equations and then the functions are replaced by their gradients. The roots of the gradients are a better approximation than the original point. This iterated equation is:

$$\hat{x}_{k+1} = \hat{x}_k - J_k^{-1} E(\hat{x}_k) \tag{9}$$

In equation (9), $\hat{x}_k$ is the estimated position of the target in kth iteration, $E(\hat{x}_k)$ is the cost function, and J_k is the Jacobean matrix $\partial E(\hat{x}_k)/\partial \hat{x}_k$.

For the application mentioned earlier, in which three reference points are required, the Jacobean matrix is:

$$J = \begin{pmatrix} 2(\hat{x} - x_1) & 2(\hat{y} - y_1) & 2(\hat{z} - z_1) \\ 2(\hat{x} - x_2) & 2(\hat{y} - y_2) & 2(\hat{z} - z_2) \\ 2(\hat{x} - x_3) & 2(\hat{y} - y_3) & 2(\hat{z} - z_3) \end{pmatrix} \tag{10}$$

Where $\hat{x}, \hat{y}, \hat{z}$ are the Cartesian coordinate of estimated position. x_k, y_k, z_k are the coordinate of reference points.

In this application, the inversion of the Jacobean matrix has been computed offline and embedded inside the tracking program to speed up the processing time. This makes 3-D tracking in real time become reality. The Newton method is quite simple, so it is easy to embed it into the sensor nodes without massive computational power consumption.

4 Automatic Lighting Control

To control the beam pointing to the known spot obtained from the positioning system, coordinate transfer and a DMX-512 controller are used.

Fig. 5 shows the coordinate transfer process. Because the intelligent fixture has two degrees of freedoms, pan and tilt, while the positioning system provides the target information with Cartesian coordinates (x,y,z), a mapping is needed to transfer the coordinates.

4.0.1 Global coordinate

As shown in Fig. 5, a global coordinate can be set up in the theater, so the position of the target is $P_t = (x_t, y_t, z_t)$

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

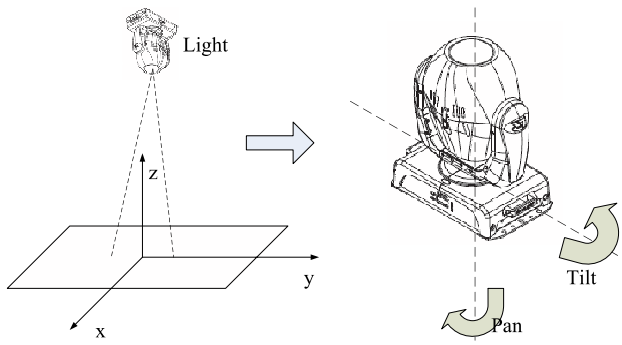


Figure 5. Coordinate Transfer

and the position of the light is $P_l = (x_l, y_l, z_l)$. Because the light position is fixed, the mapping relation is only a function of target position P_t to the Pan and Tilt.

$$Pan = \arctan \frac{x_t - x_l}{y_t - y_l} + Pan_{offset} \tag{11}$$

$$Tilt = \arctan \frac{\sqrt{(x_t - x_l)^2 + (y_t - y_l)^2}}{z_t - z_l} + Tilt_{offset} \tag{12}$$

Where the Pan_{offset} and $Tilt_{offset}$ are the offsets of the device input that makes the intelligent fixture fit the global coordinate exactly.

4.0.2 Calibration

An issue with equation (11) is that it assumes the position of the light $P_l = (x_l, y_l, z_l)$ as a known parameter. In practice, it is hard to measure the exact position of the moving lights, which are quite heavy, because they are usually fixed on the ceiling, unlike the wireless sensors which can be mounted flexibly. Calibration is an easy solution in this situation. Calibration is the process setting the magnitude of the output to the magnitude of the input property. In this application, the calibration process is defined as follows:

1. Build the global coordinate system, set up the original spot.
2. Mount the wireless sensors carefully, making sure they can get the position information correctly.
3. Fix the moving light on the roof.
4. Select three points in the theater, get the coordinates by the positioning system.
5. Control the light beam to point to the selected points manually, recording the Pan and Tilt respectively.
6. Use the coordinate of points and the direction of the moving fixture as known parameters in equation (11).
7. Solve the equations by numerical function and get the position of the light which is $P_l = (x_l, y_l, z_l)$.

4.1 DMX-512 controller

There is a special protocol used by intelligent lighting devices, namely DMX 512 [8]. DMX 512 is an entertainment industry standard based on RS485, which is mainly used to control stage lighting. Each DMX cable can support 512 channels. Thus, the controller can transmit up to 512 8-bit values at one time. Depending on the kind of device, each intelligent fixture requires 20 to 40 channels. Data are transmitted at the speed of 250kbps. There are 513 bytes in one package, which are 512 byte values of the 512 channels and 1 byte start code. Therefore, the refresh rate of the light device is 44Hz, which is enough because the mechanical delay of the intelligent light is relative large.

In our application, a USB to DMX converter is used to help the computer send and receive DMX packages.

5 System Setup

The entire system is the integration of the positioning subsystem and the lighting control subsystem. The system framework is shown in Fig. 6.

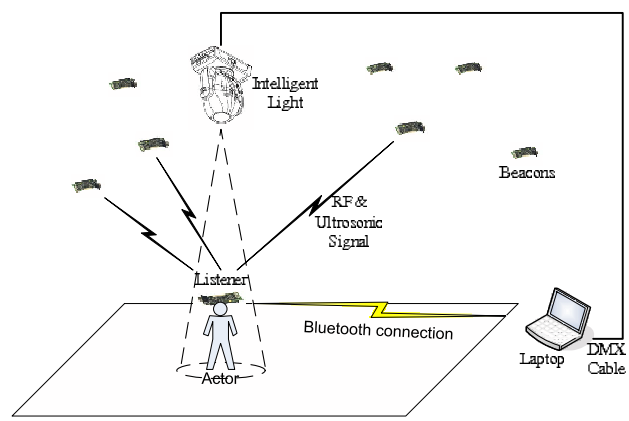


Figure 6. System Framework

5.1 Cricket sensors

Eight Cricket sensor nodes were used to build the positioning system. One of them is configured as the listener, which is carried by the actor. The other seven nodes are beacons that send RF ID and ultrasonic pulses periodically. The time interval of each transmission is pre-configured between 400 ms and 800 ms to provide enough refresh frequency with a low collision rate. The basic Cricket sensor

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

system requires the listener to connect to a computer directly through a serial cable. In our application, this is not acceptable in our application because an actor cannot carry a long cable during the performance. To solve this problem, a serial to bluetooth converter was used to build a wireless communication channel.

5.2 Highend Studio Beam

Highend Studio Spot 250 [7] is the intelligent fixture used in the automatic lighting system. The fixture can rotate $540°$ in pan and $270°$ in tilt. An optical encoder guarantees position accuracy. The fixture uses 18 DMX channels to control position, light, color, gobo, etc. During the performance, the data related to light, color and gobo can be changed remotely on the computer, and the pan and tilt data will change automatically.

6 Experiment and Performance

To validate the automatic lighting system, some experiments and performance results are presented. The listener sensor and the bluetooth converter were carried by the actor. The sensor was on the shoulder and faced up so that it could receive the ultrasonic signals without problems. The other devices were mounted on the waist, which is similar to carrying a wireless microphone. Another bluetooth receiver was plugged into the control laptop. A special hexangular frame was made by a stainless steel pipe, each side of which was 9 feet (272.7 cm). The seven beacons were mounted on the vertexes and the center of the frame. The frame was hung on the top of the theater so that the sensors covered a large round area. When the actor entered the area, the beam of the intelligent fixture tracked him or her. The experimental program GUI is shown in Fig. 7. It was written in C++ and displays the position of the target in real time.

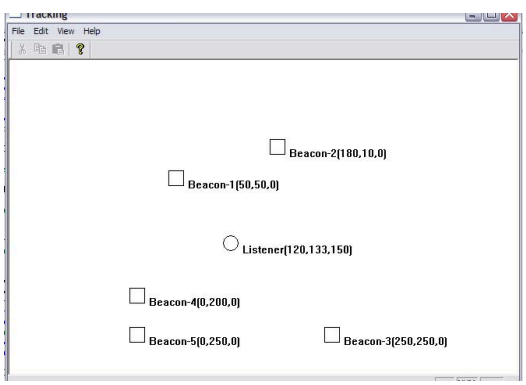


Figure 7. Tracking Program GUI

Table 1. DMX channel setup of the fixture

Channel	Description	Value
1	Pan, coarse adjustment	0-255
2	Pan, fine adjustment	0-255
3	Tilt, coarse adjustment	0-255
4	Tilt, fine adjustment	0-255
12	Shutter, close or open	0-23, 232-255
13	Dim, close or open	0, 255

6.1 Fixture setup

The bottom-up fixture was hung from the ceiling. One long DMX cable was used for communication between the fixture and the laptop. The start DMX channel number of the fixture was set to 1, so that the channels used during the performance were as in Table 1. The other channels were not used and were set to 0. As an example, if the position requirement of the fixture is that Pan equals $270°$, Tilt equals $135°$ and the beam is on, then the data of channel one and three should be set to 127; the data of channel 12 and 13 should be set to 255, and so on.

6.2 Experiment and Performance results

In the experiment, as soon as the actor carrying the listener sensor entered the sensing area, the automatic lighting system would illuminate him until he left the valid field. The experiment results show that the tracking system achieved accurate indoor wireless tracking. Tracking error was within 5 centimeters and the average system latency was 60 ms.

After several experiments, the automatic lighting system was used in the Crosstalk concert performance at the University of Arizona on May 6th, 2006, shown in Fig. 8. The performance was the first attempt ever to use an automated fixture in a real time performance.

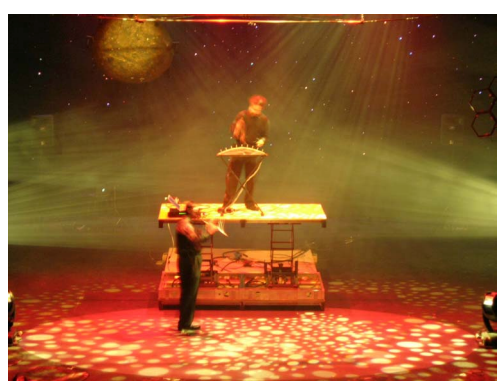


Figure 8. Crosstalk Performance

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

7 Future work

The wireless sensor network based automatic lighting controller described in this paper is the first autonomous lighting system based on wireless sensor networks used in a real performance. The performance was successful and validated the system. A drawback of the system is relatively long latency, especially when an actor is moving at a high speed. Another issue is that the sonar sensor can be blocked by obstacles causing the tracking system to lose the target. Our future work is to minimize the latency of the system and to try to utilize a more robust sensor setup.

References

[1] Cricket Project, "Cricket v2 User Manual," MIT Computer Science and Artificial Intelligence Laboratory, 2004.

[2] S. Das, C. Gleason, and et al., "2-D Tracking Performance Evaluation Using the Cricket Location-Support System," *2005 IEEE International Conference on Electro Information Technology*, pp. 1-6, 2005.

[3] R. Singh, M. Gandetto, and et al., "A novel positioning system for static location estimation employing WLAN in indoor environment," *15th IEEE International Symposium on Personal, Indoor and Mobile Radio Communications*, Vol. 3, pp. 1762 - 1766, 2004.

[4] K. Hida, M. Mizutani, and et al., "Design of Goods Tracking System with Mobile Detectors," *1st International Symposium on Wireless Pervasive Computing*, pp. 1-6, 2006.

[5] Y. Fukuju, M. Minami, and et al., "DOLPHIN: an autonomous indoor positioning system in ubiquitous computing environment," *2003. IEEE Workshop on Software Technologies for Future Embedded Systems*, pp. 53-56, 2003.

[6] M. Odam, "Theatre lighting control for Beauty and the Beast," *Power Engineering Journal*, Vol. 12, pp. 89-94, 1998.

[7] High End System, "Studio Beam User Manual," High End Systems, Inc., 2004.

[8] Enttec, `http://www.enttec.com/opendmxusb.php`, 2006

[9] Wikipedia, `http://en.wikipedia.org/wiki/Intelligent_lighting`, 2006

[10] H. Karimi, J. Lockhart, "GPS-based tracking systems for taxi cab fleet operations," *Proceedings of the IEEE-IEE Vehicle Navigation and Information Systems Conference, 1993*, pp. 679-682, 1993.

[11] Crossbow Technology, `http://www.xbow.com/wireless_home.aspx`, 2006 Crossbow Technology, Inc.

[12] P. Maycheck, "Stochastic Models, Estimation, and Control," *Academic Press*, Vol 1. Chapter 1.

Dynamic Scheduling of Skippable Periodic Tasks in Weakly-Hard Real-Time Systems

Maryline Chetto
IRCCyN – University of Nantes
1 Rue de la Noé, F-44321 Nantes FRANCE
Maryline.chetto@univ-nantes.fr

Audrey Marchand
IRCCyN – University of Nantes
1 Rue de la Noé, F-44321 Nantes FRANCE
Audrey.marchand@univ-nantes.fr

Abstract

This paper deals with dynamic scheduling in real-time systems that have Quality of Service requirements. We assume that tasks are periodic and may miss their deadlines, occasionally, as defined by the so-called Skip-Over model. In this paper, we present a dynamic scheduling algorithm, called RLP (Red as Late as possible), a variant of Earliest Deadline to make slack stealing and get better performance in terms of ratio of periodic task instances which complete before their deadline. Simulation results show that RLP outperforms the two conventional skip-over algorithms, namely RTO (Red Tasks Only) and BWP (Blue When Possible), introduced about ten years ago. Finally, we present the integration of these QoS scheduling services into CLEOPATRE[1], a free open-source library which offers selectable real-time facilities on shelves.

1. Introduction

Real-time systems are computer systems in which the correctness of the system depends not only on the logical correctness of the computations performed, but also on time factors. Real-time systems can be classified in three categories: hard, soft and weakly-hard.

In hard real-time systems, all instances must be guaranteed to complete within their deadlines. In those critical control applications, missing a deadline may cause catastrophic consequences on the controlled system.

For soft real-time systems, it is acceptable to miss some of the deadlines occasionally. It is still valuable for the system to finish the task, even if it is late.

In weakly-hard real-time systems, tasks are allowed to miss some of their deadlines, but there is no associated value if they finish after the deadline. Typical illustrating examples of systems with weakly-hard real-time requirements are multimedia systems in which it is not necessary to meet all the task deadlines as long as the deadline violations are adequately spaced.

There have been some previous approaches to the specification and design of real-time systems that tolerate occasional losses of deadlines. Hamdaoui and Ramanathan in [1] introduced the idea of (m,k)-firm deadlines to model tasks that have to meet m deadlines every k consecutive invocations. The Skip-Over model was introduced by Koren and Shasha [2] with the notion of skip factor. It is a particular case of the (m,k)-firm model. They reduce the overload by skipping some task invocations, thus exploiting skips to increase the feasible periodic load.

In this paper, we address the problem of the dynamic scheduling of periodic task sets with skip constraints. In this context, the objective of a scheduling algorithm is to maximize the effective QoS (Quality of Service) of periodic tasks defined as the number of task instances which complete before their deadline.

The remainder of this paper is organized as follows: Section 2 presents relevant background materials about the Skip-Over model. We describe two basic scheduling algorithms, namely RTO and BWP which are based on this model. In section 3, we recall the foundation of EDL (Earliest Deadline as Late as possible) algorithm, a specific method to optimize system slack by running the hard deadline tasks at the latest time while still guaranteeing their timing requirements [3]. Then, we show how to use EDL for providing an efficient scheduling algorithm called RLP (Red as Late as Possible) for the Skip-Over model. Simulation results are reported in section 4 in order to show RLP performances compared to RTO and BWP. In section 5, we provide an algorithmic

[1] work supported by the French research office, grant number 01 K 0742.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

description of the RLP scheduler. In section 6, we present the integration of these QoS scheduling services into CLEOPATRE, a free open-source library which offers selectable real-time facilities on shelves and we report measures in terms of footprint and time overheads. Section 7 summarizes our contribution and gives directions for future works.

2. Background materials

2.1. The skip-over model

In what follows, we consider the problem of scheduling periodic tasks which allow occasional deadline violations (i.e., skippable periodic tasks), on a uni-processor system. We assume that tasks can be preempted at any time and they do not have precedence constraints. A task T_i is characterized by a worst-case computation time C_i, a period P_i, a relative deadline equal to its period, and a skip parameter s_i. This parameter represents the tolerance of this task to miss deadlines. That means that the distance between two consecutive skips must be at least s_i periods. When s_i equals to infinity, no skips are allowed and T_i is a hard periodic task. So, the skip parameter can be viewed as a QoS metric (the higher s_i, the better the quality of service).

Every task T_i is divided into instances where each instance occurs during a single period of the task. Every instance of a task is either red or blue [2]. A red task instance must complete before its deadline; A blue task instance can be aborted at any time. However, if a blue instance completes successfully, the next task instance is still blue.

2.2. RTO and BWP algorithms

Two scheduling algorithms were introduced about ten years ago by Koren and Shasha [2]. Under the *Red Tasks Only* (RTO) algorithm, red instances are scheduled as soon as possible according to *Earliest Deadline First* (EDF) algorithm [4], while blue ones are always rejected.

The *Blue When Possible* (BWP) algorithm is an improvement of RTO. Indeed, BWP schedules blue instances whenever their execution does not prevent the red ones from completing within their deadlines. In other words, blue instances are served in background relatively to red instances.

2.3. Earliest Deadline as Late as possible

Let us review the fundamental properties of EDF algorithm, stated in [3] and [5] which are the basic foundation of our approach for scheduling tasks in the skip-over model. In general, implementation of EDF consists in executing tasks according to their urgency, as soon as possible with no inserted idle time. Such implementation is known as EDS (*Earliest Deadline as Soon as possible*).

Nevertheless, in some applications, this implementation presents drawbacks, for example when soft aperiodic tasks need to be served with minimal response times. In that case, it is preferable to postpone execution of periodic tasks, executing them by the so called EDL (*Earliest Deadline as Late as possible*) strategy. Such approach is known as Slack Stealing since it makes any spare processing time available as soon as possible. In doing so, it effectively steals slack from the hard deadline periodic tasks.

A means of determining the maximum amount of slack which may be stolen, without jeopardizing the hard timing constraints, is thus key to the operation of the EDL algorithm. In [3], we described how the slack available at any current time can be found. This is done by mapping out the processor schedule produced by EDL for the periodic tasks from the current time up to the end of the current hyper-period (the least common multiple of task periods). This schedule is constructed dynamically whenever necessary and is computed from a static EDL schedule which is constructed off-line and memorized by means of the two following vectors:

- K, called static deadline vector. K represents the time instants from 0 to the end of the first hyper-period, at which idle times occur and is constructed from the distinct deadlines of periodic tasks.
- D, called static idle time vector. D represents the lengths of the idle times which start at time instants of K.

The complexity for computing the EDL static schedule is O(N) where N is the total number of periodic instances in the hyperperiod.

At run time, the dynamic EDL schedule is updated from the static one by taking into account the execution of current ready tasks. It is described by means of the two following vectors:

- K_t, called dynamic deadline vector. K_t represents the time instants posterior to t in the current hyperperiod , at which idle times occur.
- D_t, called dynamic idle time vector. D_t represents the lengths of the idle times that start at time instants given by K_t.

The complexity for computing the EDL dynamic schedule is O(K.n) where n is the number of periodic tasks, and K is equal to $\lfloor R/p \rfloor$, where R and p are respectively the longest deadline and the shortest period of current ready tasks [5].

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

3. The RLP algorithm

3.1. Principles of RLP algorithm

The objective of RLP algorithm is to bring forward the execution of blue task instances so as to minimize the ratio of aborted blue instances, thus enhancing the QoS (i.e., the total number of task completions) of periodic tasks. From this perspective, RLP scheduling algorithm, which is a dynamic scheduling algorithm, is specified by the following behavior:

- if there are no blue task instances in the system, red task instances are scheduled as soon as possible according to the EDF (Earliest Deadline First) algorithm.
- if blue task instances are present in the system, they are scheduled as soon as possible according to the EDF algorithm (note that it could be according to any other heuristic), while red task instances are processed as late as possible according to the EDL algorithm. Deadline ties are always broken in favor of the task with the earliest release time.

The main idea of this approach is to take advantage of the slack of red periodic task instances. Determination of the latest start time for every red instance of the periodic task set requires preliminary construction of the schedule as described previously and taking skips into account [6]. In the EDL schedule established at time t, we assume that the instance following immediately a blue instance which is part of the current periodic instance set at time t, is red. Indeed, none of the blue task instances is guaranteed to complete within its deadline.

Moreover, in [5] it was proved that the online computation of the slack time is required only at time instants corresponding to the arrival of a instance while no other is already present on the machine. In our case, the EDL sequence is constructed not only when a blue task is released (and no other was already present) but also after a blue task completion if blue tasks remain in the system (the next task instance of the completed blue task has then to be considered as a blue one).

Note that blue tasks are executed in the idle times computed by EDL and are of same importance beside red tasks (contrary to BWP which always assigns higher priority to red tasks).

3.2. Illustrative example

To illustrate RLP, let us consider a set of five periodic tasks $T = \{T_0, T_1, T_2, T_3, T_4\}$ whose parameters (computation time and period) are described in Table 1. We assume that all the tasks have the same skip parameter $s_i = 2$. We note that the processor utilization factor i.e the processing workload for this task set is equal to 1.15 and consequently some instances will necessarily miss their deadlines. It can be observed on Figure 1 that, thanks to RLP scheduling, the number of deadline violations relative to blue task instances has been reduced to three.

T_i	T_0	T_1	T_2	T_3	T_4
C_i	3	4	1	7	2
P_i	30	20	15	12	10

Table 1 : Task parameters

They occur at time instants t = 40 (task T_4), and t = 60 (tasks T_3 and T_4). Observe that T_3 first blue task instance which would fail to complete within its deadline with the BWP strategy, has enough time to succeed in the RLP schedule, since the execution of T_1 and T_0 first red task instances is postponed.

Until time t = 10, red task instances are scheduled as soon as possible. From time t = 10 to the end of the hyper-period (defined as the least common multiple of task periods), red task instances do execute as late as possible in the presence of blue task instances, thus enhancing the QoS of periodic tasks.

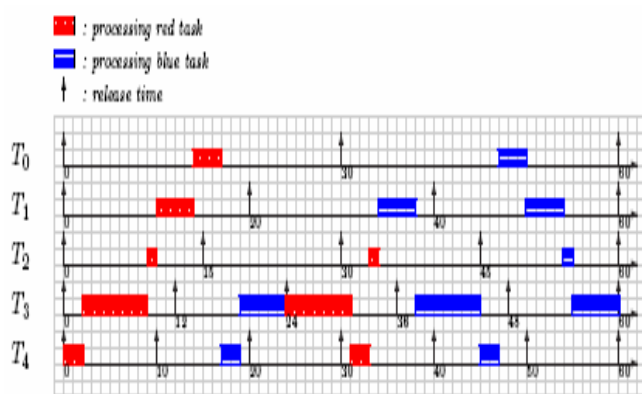


Figure 1 : A RLP schedule

4. Experimental results

4.1. Simulation parameters

The simulation context includes 50 periodic task sets, each consisting of 10 tasks with a least common multiple equal to 3360. Tasks are defined under QoS constraints with uniform s_i. Their worst-case execution time depends

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

on the setting of the periodic load Up. Deadlines are equal to the periods and greater than or equal to the computation times. Simulations have been processed over 10 hyper-periods. Measurements rely on the success ratio i.e the ratio of periodic tasks instances which complete before their deadline. The evaluation is done by varying the periodic task load, Up.

4.2. Observations

Simulation results reported in Figure 2 and Figure 3 are carried out for a skip parameter s_i equal to 2 and 6 respectively, varying the periodic load and measuring the percentage of periodic task instances that complete successfully.

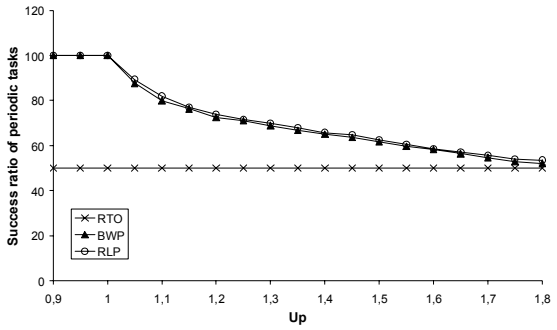


Figure 2 : QoS for uniform s_i =2

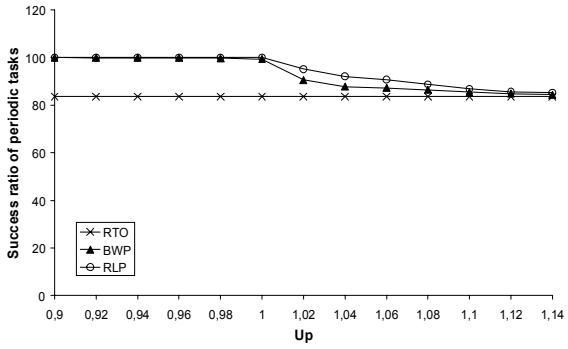


Figure 3 : QoS for uniform s_i =6

We observe that, for any skip parameter and any processor workload, BWP and RLP outperform RTO for which the resulting QoS is constant and minimal. For $U_p \leq 1$, the processor is under-loaded, and both BWP and RLP success in completing all blue tasks instances which are respectively executed after and before red task instances. In overload situations, RLP reveals better than BWP and, higher is the skip parameter more significant is the advantage of RLP over BWP.

5. Implementing the RLP scheduler

The RLP scheduler is performed by the `RLP schedule()` function, reported hereafter. In our implementation, the scheduler maintains three task lists which are sorted in increasing order of deadline: waiting list, red ready list and blue ready list.

- waiting list: list of waiting tasks.
- red ready list: list of red scheduled tasks
- blue ready list: list of blue scheduled tasks

Note that tasks in the red ready list are always performed before any one present in the blue ready list . At `RLP schedule()` invocation time, the currently running task is the default candidate to run next.

```
RLP schedule(t : current time)
begin
/*Checking blue ready list in order to
abort tasks*/
while (task=next(blue ready list)=not(Ø))
if (task→release time+task→critical
delay<t) break
endif
Pull task from blue ready list
task→release time+=task→period
task→current skipvalue=1
Put task into waiting list
 Endwhile

/*Checking waiting list in order to
release tasks*/
 while(task=next(waiting list)=not(Ø))
  if (task→release time>t) break
  endif
  if((task→current skipvalue<task→max
skipvalue) and (Slack(t)=0))
     Pull task from waiting list
     Put task into red ready list
  else
     if (blue ready list=Ø)
        Compute EDL_schedule
     endif
     if (Slack(t)!=0)
        Pull task from waiting list
        Place task into blue ready list
     endif
  endif
 endwhile

/*Checking red ready list in order to
suspend tasks*/
 while(task=next(red readylist))=not(Ø))
  if (blue ready list=not(Ø)) and (
Slack(t)!=0)
     Pull task from red ready list
     Put task into waiting list
  endif
```

```
 endwhile
end
```

The `RLP schedule()` routine proceeds in three steps. In the first one, it examines blue ready list in order to or abort one or several blue tasks which have reached their deadline. The waiting list is scanned in the second step so as to resume tasks whose release time is less than or equal to current time. Red tasks are put in the red ready list when there is no slack at current time, contrary to blue ones released only when there is an idle time.

Slack value at time t is the output of the `Slack(t)` function, obtained from the EDL schedule. Such schedule is defined by computing the length of every processor idle time which follows every task deadline in the current hyper-period. In the last step, the red ready list is examined in order to suspend red ready tasks (released before current time), provided the blue ready list is not empty and there is slack at current time i.e. surplus processing time.

6. Integration in a free operating system

6.1. The Cleopatre library

A library of free software components was developed within the French National project CLEOPATRE (Software Open Components on the Shelf for Embedded Real-Time Applications) in order to provide more efficient and better service to real-time applications. Our purpose was to enrich the real-time facilities of real-time Linux versions, such as RTLinux [7] or RTAI [8]. RTAI was the solution adopted for this project because we wanted the CLEOPATRE components to be distributed under the LGPL license which is also the one used in the RTAI project.

The CLEOPATRE library offers selectable COTS (Commercial-Off-The-Shelf) components dedicated to dynamic scheduling, aperiodic task service, resource control access, fault-tolerance and now, QoS scheduling (see Figure 4). An additional layer named TCL (Task Control Layer) interfaces all the CLEOPATRE components. It has been added as a dynamic module in $RTAI DIR/modules/TCL.o, and represents an enhancement of the legacy RTAI scheduler defined in $RTAI DIR/modules/rt sched.o. CLEOPATRE applications are highly portable to any new CPU architecture thanks to this OS abstraction layer which makes the library of services, generic. The CLEOPATRE Off-the-Shelf components are optional except the OS abstraction layer (TCL) and the scheduler.

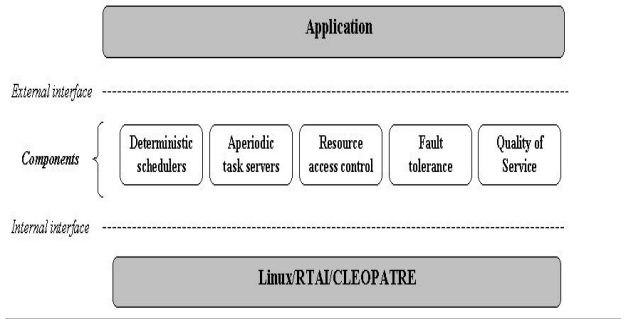


Figure 4 : The CLEOPATRE framework

At most one component per shelf can be selected. Since all components of a given shelf have the same programming interface, they are interchangeable. Everything needed to use and develop CLEOPATRE can be downloaded from the web site of the project: http://cleopatre.rts-software.org.

RTO, BWP and RLP algorithms have been put into an additional shelf called Quality of Service. The QoS services are available as independent software components. This enables developers to build their own application-specific operating system.

6.2. Data structures and API

The basic data structure of our QoS schedulers is the task descriptor, defined in $RTAI DIR/include/QoS.h as `struct QoSTaskStruct`.

This one contains nine fields for every task gathered and described in the following data structure:

```
typedef struct QoSTaskStruct
QoSTaskType;
struct QoSTaskStruct{
void (*fct) (QoSTaskType *);
/*pointer to task function*/
TaskType TCL task;
/*low-level descriptor*/
TimeType critical delay; /*deadline*/
TimeType period;
TimeType release time;
unsigned int max skipvalue;
/*maximum tolerance to skips*/
unsigned int current skipvalue;
/*dynamic skip parameter*/
unsigned int current shift;
/*shift compared with a RTO sequence*/
unsigned int slack;
/* slack time of the task*/ };
```

At initialization time, the user has to set the usual parameters for all tasks (period P_i, critical delay d_i,...) and also the additional skip parameter s_i for all QoS tasks. The user interface for the QoS schedulers is composed of the following functions:

```
QoS create     : create a new task
QoS resume     : resume a task
QoS wait       : wait till next period
QoS delete     : delete a task
```

6.3. Overheads and footprints

For embedded real-time applications, the memory footprint and disk footprint of the operating system are generally key issues as well as the time overhead incurred by its execution. Measurements of footprints for the schedulers have been realized with a Linux based real-time operating system, namely CLEOPATRE presented hereafter. Results are given in Table 2.

Scheduler	Hard Disk size (Kb)	Memory size (Kb)
RTO	3.2	2.3
BWP	4.1	3.2
RLP	9.7	7.6

Table 2 : Footprints

Table 2 shows that a scheduler, when loaded in memory (by the *insmod* command) has a size approximately 33% lower than that on the hard disk. This phenomenon is mainly due to information which is contained on the disk and used by Linux to dynamically assemble the interfaces of the software components at the loading stage.

We observe that RLP requires less than ten Kb. We have made some experiments to get a quantitative evaluation about the overhead introduced by the RLP scheduler. The tests have consisted in measuring the overhead for different number of tasks (5, 10, 15, 20,...) with all periods equal to 10 milliseconds. Periods of all tasks are harmonic, leading up to an hyper-period equal to 3360 ticks. Measurements were performed over a period of 1000 seconds on a computer system with a 400 MHz Pentium II processor with 384 Mo RAM. Results are shown in Figure 4.

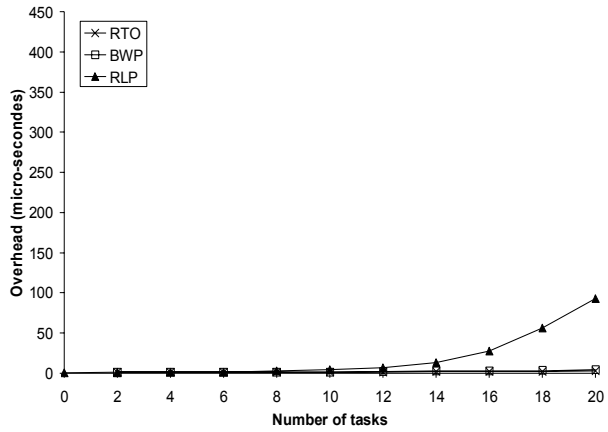


Figure 4 : Overheads of RTO, BWP and RLP

The timings shown hereafter were performed with a 1,7GHz Pentium 4 by using the Time Stamp Counter (TSC) available with every modern Intel processor. As it can been seen from Figure 4, the overhead of the QoS schedulers scales with the number of installed tasks. We note that BWP mean execution time is quite higher than the one observed for RTO. This is caused by the blue task management performed under BWP. The curve obtained for RLP is mainly due to the amount of time spent on the EDL schedule (performed only when a blue task instance is released or completed). As a matter of fact, we observe that overheads are closely related to algorithm efficiencies. An interesting feature of this component approach is that the selected scheduler can be tuned to balance performance versus complexity, so easily conforming to applications requirements.

6.4. A programming example

Success of real-time systems comes from both ease of use and performances. Writing code to run with CLEOPATRE is as simple as writing a C language program to run under Linux. The scheduling of QoS tasks is performed in the QoS schedule() function as described in §5. The scheduling occurs on timer handler activation (each 8254 interrupt).

The following example brings to light some of the features of CLEOPATRE and shows the different steps in developing under this environment. The program implemented under Cleopatre is described below:

```
/*—---Headers of all components—*/
#include <TCL.h>
#include <QoS.h>
#include <simul.h>

/*———Timer clock period (10ms)——*/
```

```
#define TIMERTICKS 10000000

/*———-Declaration of QoS tasks———*/
QoSTaskType T1;
QoSTaskType T2;

/*—Code description of QoS tasks—-*/
void CodeT1() {simul.wait(4);}
void CodeT2() {simul.wait(1);}

/*——— initialization———------*/
int init module(void)
{
TCLCreateType create={0, 2000, 0, 0};
/*********initializing QoS
tasks******/
QoS.create(&T1, CodeT1, 4, 20, 20, 2,
create);
QoS.create(&T2, CodeT2, 1, 15, 15, 2,
create);
QoS.resume(&T1,100);
QoS.resume(&T2,100);

/***starting the real-time mode***/
TCL.begin(TIMERTICKS, 20000);
return 0;}

/* ending and deleting QoS tasks */
void cleanup module(void)
{TCL.end();}
```

The application, which is only an academic one for simplicity considers a periodic task set T composed of two tasks both with a skip factor that equals 2. In that example period coincides with deadline and is 20 and 15 for T_1 and T_2 respectively.

7. Concluding remarks and extensions

The paper has described scheduling algorithms dedicated to uni-processor systems that may experience overload. We have considered the Skip-over model where all tasks are periodic and characterized by a skip factor. Because using specific properties of Earliest Deadline, we have shown that the so-called RLP algorithm performs better than other skip-over strategies when the resulting QoS is measured in terms of global success ratio.

Recently, this approach was extended to cope with aperiodic tasks that arrive at unpredictable times [6]. Aperiodic tasks may have strict deadline or no deadline at all. The objective of the skip-over scheduler is then to serve all the tasks by accounting for both timing and QoS constraints.

In this paper, the order used by the scheduler for executing the blue instances is based on the relative deadline but could be selected based on a particular performance metric. We are no studying new strategies for achieving fairness while maximizing QoS [9]. We have compared different methods for ordering ready blue instances, aiming to balance individual success ratios for fairness motivations.

Finally, we have integrated these QoS functionalities in CLEOPATRE which is a portable, open-source, free to download and royalty free RTOS that can be used in commercial applications through the LPGL license.

8. References

[1] M. Hamdaoui, P. Ramanathan, A Dynamic Priority Assignment Technique for Streams with (m,k)-firm deadlines. *IEEE Transactions on Computers*, Vol. 44, No. 4, pp. 1443-1451, 1995.

[2] G. Koren., D. Shasha, An Approach To Handling Overloaded Systems That Allow Skips. *Proceedings of the 16th IEEE Real-Time Systems Symposium (RTSS'95)*, Pisa, Italy, pp. 110-119, 1995.

[3] H. Chetto, M.Chetto., Some Results of the Earliest Deadline Scheduling Algorithm. *IEEE Transactions on Software Engineering,* Vol. 15, No. 10, pp. 1261-1269, 1989.

[4] C. Liu, J.W. Layland Scheduling algorithms for multiprogramming in a hard real-time environment .*Journal of ACM*, vol.20 n°1 pp. 46-61, 1973.

[5] M. Silly-Chetto, The EDL Server for Scheduling Periodic and Soft Aperiodic Tasks with Resource Constraints, *Journal of Real-Time Systems*, Kluwer Academic Publishers, Vol. 17, pp. 1-25, 1999.

[6] A. Marchand, M. Silly-Chetto,. Dynamic Real-Time Scheduling of Firm Periodic Tasks with Hard and Soft Aperiodic Tasks. *Journal of Real-Time Systems*. Vol. 32, No.1, pp. 21-47, 2006.

[7] P. Mantegazza, DIAPM RTAI for Linux: Why's, what's and how's, Real Time Linux Workshop, University de Technology of Vienna, 1999.

[8] V. Yodaiken, the RTLinux Approach to Real-Time – FSM Labs Inc., August 2004.

[9] A.Marchand and M. Chetto, Stability and Robustsness Issues in Scheduling Periodic Tasks with Firm Real-Time Requirements, in Proc. Euromicro Conference on Real-Time Systems, WIP Session, Dresden, pp. 9-12, July 2006.

Time- and Space-Efficient Evaluation of Sparse Boolean Functions in Embedded Software

Václav Dvořák
Brno University of Technology
dvorak@fit.vutbr.cz

Abstract

The paper addresses software implementation of large sparse systems of Boolean functions. Fast evaluation of such functions with the smallest memory consumption is often required in embedded systems. A new heuristic method of obtaining compact representation of sparse Boolean functions in a form of linked tables is described that can be used for BDD minimization as well. Evaluation of Boolean functions reduces to multiple indirect memory accesses. The method is compared to other techniques like a walk through a BDD or a list search and is illustrated on examples. The presented method is flexible in making trade-offs between performance and memory consumption and may be thus useful for embedded microprocessor or microcontroller software.

1. Introduction

Efficient evaluation of Boolean functions is an important part of many embedded software systems. Some applications include: search and optimization, modeling, simulation and verification of digital circuits in CAD, extracting routing information from packets, etc. We will restrict ourselves to sparse systems of Boolean functions, as these are most frequent in practice. Also we will address Boolean functions of large numbers (tens) of variables because small size systems can be implemented directly in hardware, e.g. in PLA, ROM or TCAM (Ternary Content Addressable Memory).

Software implementation of Boolean functions will be assumed in a form of a data structure describing the function and of a compiled program that reads the input vector and evaluates the function with the use of this data structure. The size of the code and of the data structure is one figure of merit, the other is the evaluation time from reading the input to generating the output. Sometimes the evaluation time per one input may get reduced if many inputs follow one another.

Hereafter we will use three compact representations: a TCAM-like table, linked tables and binary decision diagrams (BDDs). The BDDs are well known, especially the reduced ordered BDDs (ROBDDs), [1]. On the base of ROBDDs we will develop a more practical representation - linked tables.

Software implementation of Boolean functions has been up to now studied especially in connection with PLCs ("ladder diagrams") [2], digital system simulation, formal verification and testing [1], or specialized event processing [3], where either a speed (PLC) or a required memory were not that important. On the contrary, in embedded systems we do care for performance and memory space as well as for power consumption. We will demonstrate that presently used algorithms are generally too slow (TCAM emulation, BDDs); the use of linked tables enables faster evaluation with similar (selectable) size data structures which can be minimized by utilizing don't cares directly.

The paper is structured as follows. In the following Section 2 we define sparse Boolean functions and show their description by data structures, a relation among various representations and their complexity. Our novel heuristic approach for minimizing the relevant data structures is explained in Section 3. In Section 4 we exemplify creation of data structures on a sample Boolean function and give ways how to trade speed of evaluation against required memory space in Section 5 and 6. Results obtained with selected functions of 8 to 64 variables are commented on in Conclusions.

2. Representation of sparse systems of Boolean functions

To begin our discussion, we define the following terminology. Multiple-output Boolean functions of n Boolean variables will be simply referred to as Boolean functions F_n with output values from

$$Z_R = \{0, 1, 2, \ldots, R\text{-}1\},$$

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

$$F_n : Z_2^n \rightarrow Z_R . \quad (1)$$

Under a *sparse Boolean function* we will understand function F_n: $Z_2^n \rightarrow Z_R$ with the domain split into two parts X and D, $Z_2^n = X \cup D$, $|X| << 2^n$, and specified in one of two ways:

1) $Z_2^n \setminus X = D$ is the don't care set (F_n is an incomplete function in Z_2^n, F_n: $X \rightarrow Z_R$)

2) Mapping of the don't care set $Z_2^n \setminus X = D$ into Z_R is artificially defined to make implementation as easy as possible, F_n: $X \cup D \rightarrow Z_R$.

A binary decision diagram (BDD) is a directed acyclic graph in which each decision node is labeled by a control variable tested in this node. Two edges coming out from the decision node, leading to the nodes in the subsequent levels, correspond to the values of control variable. Beside decision nodes there are terminal nodes (leaves) labeled by the value of the function that is being evaluated by the diagram. A BDD is ordered, if the order of control variables tested along every path in the BDD is the same. All the nodes of the ordered BDD (OBDD) labeled by the same variable make up a level of this diagram. An ordered BDD is reduced (ROBDD) if [1]

1. no two distinct nodes have the same control variable name and the same 0- and 1-successor node.
2. no node has the identical successor for both values of a control variable.

ROBDD is canonical (unique) representation for any given function [1]. Any pair of functions will have different ROBDDs unless the functions themselves are equivalent. In the rest of the paper, we will consider OBDDs or ROBDDs , even if the term BDD is used.

An important parameter is a size of BDD, i.e. the total number of decision nodes, as it determines the size of data structure needed to store a BDD. The construction of minimum-size BDDs belongs among NP-hard problems [9]. Upper bounds on the OBDD's size for general Boolean functions are not too encouraging, but many practical functions do have a reasonable BDD size. The upper bound on the size of the related BDD belonging to a function with L literals in its DNF can be obtained as [4]

$$\min_k \left\{ L - \left\lceil k \tfrac{L}{n} \right\rceil + 2^k - 1 \right\}, \quad k = 1,2,...,L \quad (2)$$

As the first approximation to (2) we can use L only.

Traditional description of a Boolean function by the Boolean expression (expressions in case of multiple outputs) may sometimes be the best for evaluation purposes, especially if the function of many variables is defined only in a few points. Another possibility is a list of ternary input vectors (composed of 0, 1 and don't care elements) that can specify a relevant output value for each input vector in the list. Ternary vectors can be stored as two binary vectors, the vector of values and a mask indicating don´t cares.

Efficiency of evaluation techniques strongly depends on the used description of the Boolean function. Let us illustrate above possibilities on the N queens problem. We are to compare possible representations of a Boolean function of $N \times N$ variables that returns 1 for every configuration of 1's (queens) on the $N \times N$ chessboard, such that no queen attacks another one. For example if $N = 4$, there are 2 solutions that can be generated by Boolean function F_{16} (used notation is required by the applet [5], but variables in the diagram are enumerated from 0 to 15):

$$F_{16} = !a_{11}*!a_{12}*a_{13}*!a_{14}*a_{21}*!a_{22}*!a_{23}*!a_{24}*!a_{31}*!a_{32}*!a_{33}*a_{34}*!a_{41}*a_{42}*!a_{43}*!a_{44} + !a_{11}*a_{12}*!a_{13}*!a_{14}*!a_{21}*!a_{22}*!a_{23}*a_{24}*a_{31}*!a_{32}*!a_{33}*!a_{34}*!a_{41}*!a_{42}*a_{43}*!a_{44} \quad (3)$$

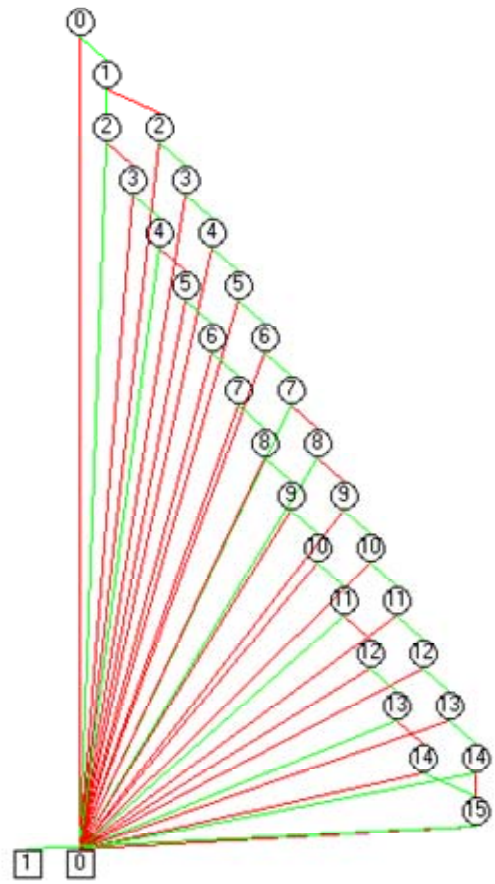


Fig.1. ROBDD for 4 queens problem

In our case L = 32, n = 16 and the upper bound according to eq. (2) is 31 nodes. The real ROBDD generated by the applet [5] has 29 nodes and is shown in Fig.1. The most efficient representation and evaluation of F_{16} is thus clear: two memory words are

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

sufficient to store two min-terms in (3) and two bitwise comparisons will do for the quickest evaluation. A BDD representation is no good in this case.

Now if we move to 8 queens problem, there will be 92 solutions described by F_{64}. Solutions can be found by the known algorithm [6], [1]. To store 92 binary vectors of length 64 is still acceptable, but instead of linear search we can order solutions and do better with the logarithmic search in $\lceil \log_2 92 \rceil = 7$ steps at most.

The BDD size is upper-bounded by 5535 nodes according to eq. (2), so that storing of such BDD would not be space efficient at all. A pass through this BDD would need 64 steps in the worst case, what is bad as well.

3. Heuristic construction of BDDs and linked tables

We did not see too much use for BDDs in the former example. However, in this section we will give a method of construction of linked tables which are much more useful for the purpose of function evaluation. The method is based on bottom-up heuristic construction of BDDs, which uses a concept of sub-function [7].

Informally, the sub-function f of F_n is a function of s variables obtained from F_n by setting $n-s$ variables to fixed constant values. The number of distinct sub-functions of s variables, s = 1, 2, …, n-1, characterizes the Boolean function and its complexity. Sub-functions themselves may also be incomplete (don't care values for some binary s-tuples). A compatibility relation can be defined on the co-domain of such sub-functions: don't care (denoted by "x") is compatible with any value from Z_R.

Using the concept of sub-functions, we will now decompose iteratively the given sparse function of 8 variables, see Fig.2. The map of the function at the top is sparsely populated by 16 function values (0 to F). For clarity don't care cells are left empty in tables, but otherwise are denoted by symbol "x" in the text. Single-variable sub-functions can be created with respect to any variable. E.g. two vertically adjacent cells correspond to a sub-function of the first variable that attains alternate values 0 and 1 at even and odd rows (see e.g. [F,8] in Fig.2). Using compatibility relation we can combine pairs [α,x] and [x,β] into a single sub-function [α,β]. Altogether nine sub-functions of the first variable are detected in the topmost table. The first decomposition step is described below the table; each sub-function is given a new symbol ([1,0]→0, [2,7]→1,[F,8]→3, etc.), thereby

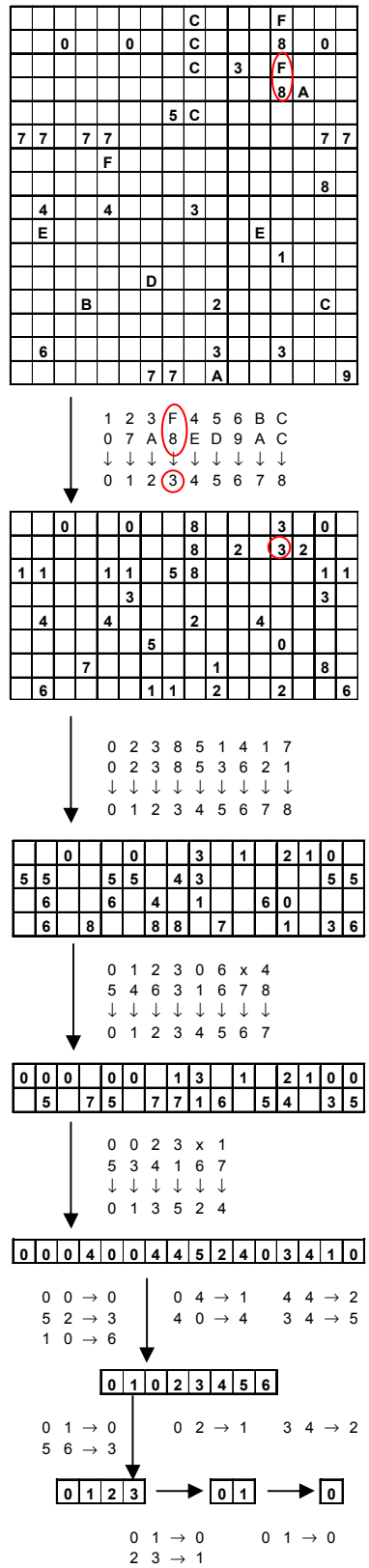


Fig. 2. Iterative decomposition (8 variables)

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

removing the first variable from the function. A map of the new intermediate Boolean function of 7 variables is now created replacing sub-functions by new values (symbols, IDs). This process repeats 8 times.

The OBDD can now be created starting from root 0. Every assignment [a,b]→ c, when reversed, specifies one decision node with input c and two outputs a and b controlled by the relevant variable. Assignments of the type [a,a]→b, [a,x]→c, [x,a]→d do not represent decision nodes because the outputs are the same (or compatible); such a decision node degenerates to a wire. Going up from the root (a map of 0 variables) to the original map of 8 variables, the OBDD in Fig.3 is created. Usually BDDs have a root at the top, but we displayed the BDD upside down in order to keep the BDD structure in correlation with the sequence of map transformations in Fig.2. Nodes are labeled by intermediate function values. Out of 46 assignments 34 correspond to decision nodes and 12 to wires only.

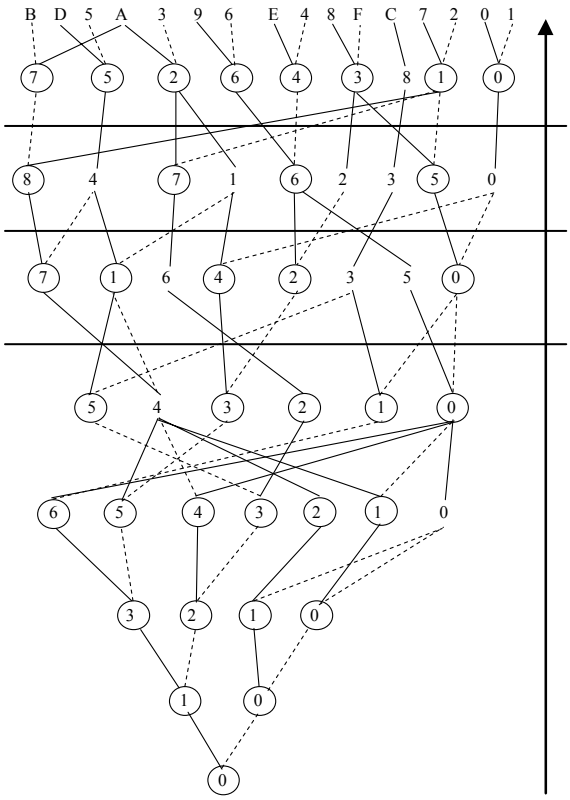


Fig.3. OBDD of the sample function of 8 variables (0 = - - - - - , 1 = ———)

In our example we did not care about variable ordering; the ordering was chosen more or less randomly. However, it is known, that the size of the BDD is determined both by the function being represented and the chosen ordering of the variables [8]. For some functions, the size of a BDD may vary between a linear to an exponential range depending upon the ordering of the variables. The problem of optimal variable ordering is unfortunately NP hard [9].

If we want to minimize the size of a BDD, the following heuristics can be used: do sub-function counting for all variables in each decomposition step and use for this step the variable with the minimum sub-function count. By intuition, the minimum count of symbols may hopefully produce a minimum count of their pairs.

Note also that the above small example with maps of the original and intermediate functions was done only for illustration. When we have sparse functions with several tens of variables represented by a list of defined points, all the processing is done on these lists. The case with don´t cares already defined for the purpose of minimization is given in Section 6.

4. Linked tables and OBDDs

In this Section we first introduce the technique of linked tables, programs for interpreting OBDDs as well as linked tables and then compare both techniques on examples.

Linked tables and OBDD are equivalent descriptions of a Boolean function; one layer of the OBDD or more layers combined can be described by a table. For example linked table 4 is constructed (Fig. 4) from the top layer of the OBDD in Fig. 3. Transformation of 9 symbols to 16 symbols is described by reversed assignments under the topmost map in Fig.2.

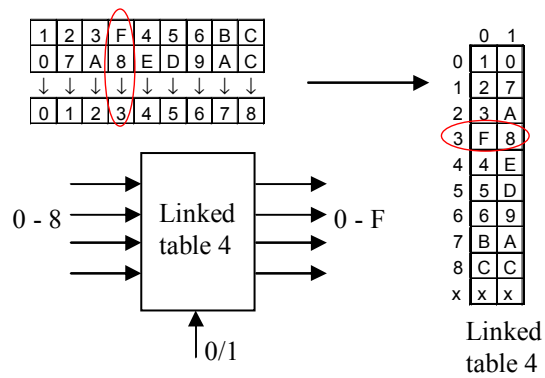


Fig.4. Construction of one of linked tables

The whole BDD (Fig. 3) is then described by 4 tables as shown in Fig. 5. The chain of tables is homogeneous, but generally the tables may have different size. However, sparse functions are typically implementable by homogeneous cascades, since the

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

number of sub-functions (and therefore decision nodes) follows a pattern: rising – constant – dropping, [4].

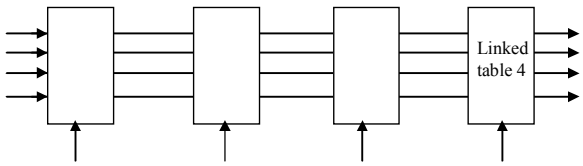


Fig.5. A chain of linked tables for the Boolean function at Fig. 2.

As can be seen, the difference between OBDD and linked tables is in communication among the layers or tables: in OBDD each symbol requires an individual edge (”wire”), whereas the symbols being sent between tables are binary coded. Another way to look at linked tables is to consider each table as an M-ary decision node and the chain of tables as a special “*in-line*” decision diagram.

This difference of two representations reflects itself in the way how the program interprets a certain application-specific OBDD or a table chain. In case of the OBDD we may use for each node a record with 3 fields. A format indicator is one-bit field specifying the leaf node. Two other fields of the leaf node are then used for output. If the node is not a leaf, two fields (adjacent words) contain pointers to the base addresses of other nodes. The base address is modified by the value of a current control variable(s) and is used to extract the correct field with the pointer to the next node. The program walks through a certain path in the ROBDD from the root to a leaf in at most *n* steps.

Linked tables are interpreted similarly, only the pointer to the next table is obtained from the current output by concatenating it with the control variable value and adding to the next table base address. As seen from Fig.5, only few steps will do. If suitable, some linked tables can be combined to provide even faster access. E.g. 4 tables in Fig.5 can be reduced to two with 6 inputs each.

5. Linked tables versus other methods

On the example of 4 queens and 16 queens we have already seen that Boolean expressions may support very fast evaluation and take up minimum memory space.

Let us now analyze 4 functions of 8 variables in Fig.2. Had we used an ordered list of defined points with function values, there would be 39 items, 8 (input) + 4 bits (output) per item, 468 bits in total, i.e. half of the full function table with size 256×4 bits. To look up the item in the table we would need $\lceil \log 39 \rceil =$ 6 steps in the worst case.

On the other hand, if we use a chain of linked tables according to Fig.5, the capacity of all tables will be $4\times(32\times4) = 512$ bits and only 4 steps (composed of read, append a value of a selected variable, add to the base address of the table to create a pointer) will do. This seems to be the best in speed and memory efficiency. Four tables may be implemented in memory as one table 32×16 bit with the correct output extracted from 16-bit word as needed. Additional flexibility is obtained with linked tables as they are combined together. For example with 2 tables 64×4 bits, the response will be 2-times faster. The size of 2 and 4 linked tables remains the same, but 2 tables combined need 64 words in memory, 8 bits per word.

As the last example we shall consider the following sparse Boolean function of 32 variables: it attains the value 1 if the given 6-bit string is detected anywhere within an input string of 32 Boolean values; otherwise the function has the value 0.

As the string of 6 consecutive values of variables may be located in 27 positions (we do not assume that the pattern wraps around), we can specify the function by 27 words of 32 ternary digits (0, 1, x). The logarithmic search is now not possible and we have to step through these words sequentially. In the worst case it may take 27 steps.

We can do much faster with linked tables, though. First the ROBDD of this function may be obtained using the applet [4], since the Boolean expression with 27 min-terms, each with 6 literals, is easy to write. The ROBDD is too large (162 nodes in total) to display, but its shape can be described like this (from the root): the number of decision nodes per level linearly increases from 1 to 6, then stays at 6 for 22 levels and finally drops from 6 to 1. From this shape of the ROBDD an optimal size and count of linked tables can be determined, Fig.6. We can keep 6 tables in 256 words of memory, 3+3+3+3+2+1= 15 bits per word. The table items can also have 1-bit format indicator “continue/end” (6 additional bits in total) and the length of processing may vary between 1 to 6 steps.

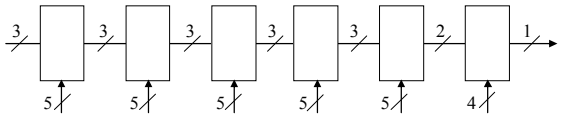


Fig. 6. Linked tables detecting 6-bit string in 32-bits

On the other hand, we could use the ROBDD implementation directly. Since there are 162 nodes, 8-bit address is needed. With format indicator (1 bit) we

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

will not be able to map one decision node to a single 16-bit word. Anyway, we can use 2 × 162 = 324 words, 16 bit each or 162 words, 32 bit each. The pass through the ROBDD may take from 1 to 32 steps. Apparently, this solution is worse than the linked tables.

Returning to the first example in Fig.1, we can also use linked tables here. Two sub-function symbols plus two constants 0 and 1 are transferred between BDD layers, so that 2-bit code will do. Possible configurations of linked tables are in Fig.7.

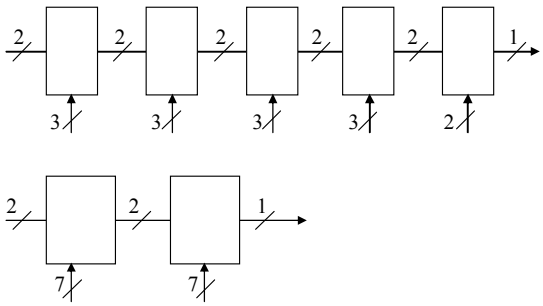


Fig. 7. Linked tables for 4 queens problem

Two table look-ups are sufficient in the shorter version, the same speed as with two comparisons suggested earlier. However, memory consumption is worse, 512 × (2+1) bits is incomparable to 2 words, 32 bits each.

6. A case study - MCS-51 microcontroller family: PLA1 and PLA2 in software

Space and time efficiency of various configurations of linked tables obtained by computer-aided iterative decomposition have been tested on two PLAs used in the core of MCS-51 family of microcontrollers,

PLA: $X \rightarrow R, X \subset Z_2^n, R \subset Z_2^r$,

with parameters in the following Table 1.

Table 1. Parameters of PLA1 and PLA2

	n	r	p	|X|	size [B]
PLA1	13	8	31	175	8192
PLA2	11	8	53	632	2048

Both PLAs implement sparse (incomplete) Boolean functions, which are after minimization described by Boolean expression in Appendix. The number of terms in AND arrays are p = 31 and 53. The size in bytes The size in bytes gives memory space $r2^n$ required for storing full function tables.

Iterative decomposition used the selection of those two variables at a time that produced the minimum number of sub-functions. Not too large size of the problem allowed still an exhaustive search – on the Pentium-based PC it took tens of seconds. The PLA1 was implemented by the cascade of 6 cells, Fig. 8a, with the total size of cell tables (ROMs) only 1792 bits. That is reduction by factor of 36. The size of tables is not uniform and evaluation would take 6 table look-ups. We can make it faster and more uniform by combining 6 cells into 3 as shown in Fig.8b. All sub-functions are counted (results given in {integer}), coded and communicated between cells, so that function values are outputs from the last cell only. The total size of linked (cell) tables is then 2816 bits; if the size of computer word *w* is known, further optimization can be done to minimize the total memory space in bytes occupied by all 3 (or possibly 4) tables.

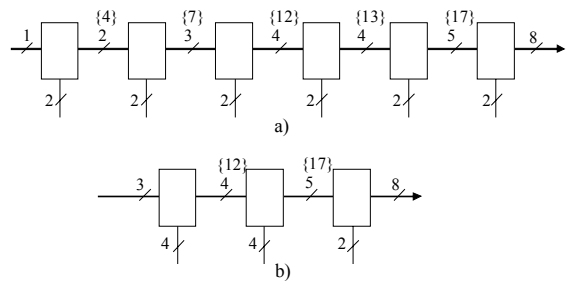


Fig. 8. Two cellular cascade implementations of PLA1

As far as PLA2 is concerned, computer-generated cascades are shown in Fig. 9. The cascade at Fig. 9b is obtained from the cascade a) by merging first two cells. The capacity of linked tables is 3264 and 3456 bits, respectively. The evaluation speed is given by 4 or 3 table look-ups.

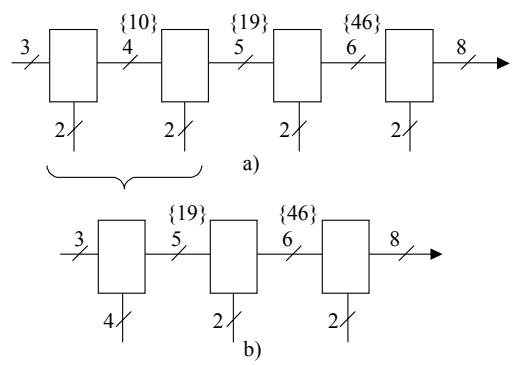


Fig. 9. Cascade of 4 or 3 cells for PLA2

We can also split output variables into two halves and then decompose them separately. The result for PLA2 is shown at Fig. 10. The size of linked tables is reduced to 1200 bits only, but the speed is reduced also. Eight table look-ups are needed and can be done

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

on one CPU core in 8 steps sequentially or on a 2-core processor concurrently in 4 steps.

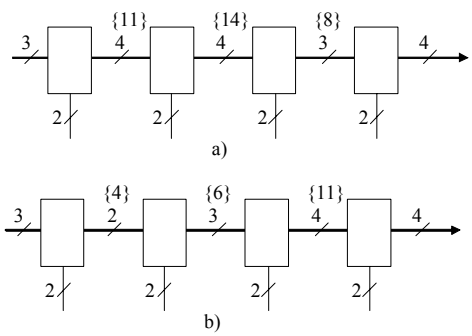


Fig. 10. Two parallel cascades implementing PLA2.

The case study of PLA1 and PLA2 offered the size of data structures and speed of evaluation as given in Table 2. The data in the table are valid under the assumptions:

- size is in bits, the length of a computer word is not considered;
- steps may have different duration in the left and the right part of the table (mask load + bitwise logical operation vs table look-ups).

Table 2. Software implementations of PLA1 and PLA2

	PLA emulation AND + OR matrix		linked tables	
	size bits	steps	size bits	steps
PLA1	1054	13 + 8	1792	6
PLA1	1054	31 + 8	2816	3
PLA2	1590	11 + 8	3456	3
PLA2	1590	53 + 8	1200	8

7. Conclusions

There is no single software evaluation method optimal for all Boolean functions. Complexity of functions that can appear in embedded systems varies a great deal and so do their space and time requirements in various evaluation techniques.

Even though the very narrow analysis done above cannot be taken as convincing, certain conclusions for engineering practice can be drawn from it, if the fast and memory efficient evaluation of sparse Boolean functions $F_n : X \rightarrow Z_R$ of several tens of variables is the main concern.

1. If the set $X \subset Z_2^n$ contains only a small number of elements, e.g. when the function is specified by DNF with few tens of minterms, the search in the ordered list of minterms can be very effective solution.

2. If $X \subset \{0,1,x\}^n$, sequential TCAM emulation may be too slow as it takes | X | steps.

3. OBDDs or ROBDDs may be useful for checking equivalence between two implementations or for formal verification [1], but they are less useful for evaluation purposes in both speed as well as memory consumption.

4. Linked tables obtained from ROBDDs seem to be a very good and effective data structure and should always be considered for evaluation of Boolean functions. They are flexible in making trade-offs between response time and memory consumption. If implemented as special hardware (a cascade of ROMs), they can support pipeline processing with one evaluation in each ROM cycle. Otherwise, in case of software implementation, several linked tables can be compacted into one table and stored in memory. The evaluation then reduces to a short chain of indirect memory accesses. Generally speaking, every sparse function can be implemented as a chain of linked tables or equivalently as a special "in-line" multi-valued decision diagram [4].

Future research will be oriented to study of evolutionary techniques for iterative decomposition of sparse Boolean functions of many variables where the exhaustive search is out of question. Large systems specified by expressions (such as those in Appendix) will be tackled either by parallel processing or by hardware acceleration.

8. References

[1] H.R.Andersen, Lecture notes for 49285 Advanced Algorithms E97, www.itu.dk/~hra/notes-index.html

[2] F. D. Petruzella: *Programmable Logic Controllers,* McGraw Hill Science/Engineering/Math, 2004.

[3] R. Sosic, J. Gu, and R. Johnson. "The Unison algorithm: Fast evaluation of Boolean expressions". *ACM Transactions on Design Automation of Electronic Systems*, 1(4): pp. 456--477, Oct. 1996.

[4] V. Dvořák, "Bounds on Size of Decision Diagrams", JUCS - *The Journal of Universal Computer Science*, Berlin, Heidelberg, CZ, Springer, 1997, s. 2-22.

[5] University of Hamburg, Department of Informatics, http://tams-www.informatik.uni-hamburg.de/applets

[6] W. Stallings, *Computer Organization and Architecture*, Sixth Edition, Prentice Hall, 2005.

[7] I. Wegener, *The Complexity of Boolean Functions*. John Wiley & Sons, New York, 1987.

[8] R. Drechsler, B. Becker, *Binary Decision Diagrams - Theory and Implementation.* Springer 1998.

[9] B. Bollig, I. Wegener, "Improving the Variable Ordering of OBDDs Is NP-Complete". *IEEE Transactions on Computers*, 45(9):993—1002, September 1996.

Acknowledgement

This research has been carried out under the financial support of the research grants "**Design and hardware implementation of a patent-invention machine**", GACR 102/07/0850, Grant Agency of Czech Republic, 2007-2009 and "Security-Oriented Research in Information Technology", MSM 0021630528..

Appendix

Programmable logic arrays PLA1 and PLA2 in MCS-51 microcontroller family

Legend: ! = logical negation, * = logical AND,
+ = logical OR

PLA1
Inputs: A, B, C, D, E, F, G, H, I, J, K, L, M
Outputs: SO, CS, BL, NL, V1, V3, V4, V5

SO = !A*!G*!I*J*M+A*!B*!I*J*M+A*F*!I*M

CS = !A*!B*D*!E*!F*!G*!H*!I*!K*!L*M + A*B*!E*!F*!G*!H*!I*!J!K*!L*!M + !A*!E*!I*M + !E*!I*J*M+!D*!I*M

BL = !B*E*!F*!G*!H*!I*!J*!K*!L + !B*C*!D*!H*!I*!J*M + !B*D*E*!H*!I*!J*M + !D*!I*!J*K*M + !A*!G*!I*J*M + E*H*!I*!L*M +C*!D*G*!I*M + !A*F*!I*M + G*!I*K*M + E*G*!I*M

NL = !B*E*!F*!G*!H*!I*!J*!K*!L + C*!D*!H*!I*L*M + !D*!I*!J*K*M + !A*!G*!I*J*M + D*E*!N*!I*M + !A*F*!I*M + E*!I*!L*M + G*!I*K*M

V1 = !A*!G*!I*J*M + C*!D*F*!I*M + A*!B*!I*J*M + !A*F*!I*M + F*!I*K*M + E*F*!I*M

V3 = !B*!C*!D*E*!F*!G*!H*!I*!J*!K*!L + !B*!G*!I*J*K*M + !D*!I*!J*K*M + B*C*!I*K*M

V4 = !B*C*!D*E*!F*!G*!H*!I*!J*!K*!L + !B*D*E*!F*!G*!H*I*!J*!K*!L*M + !A*!G*!I*J*L*M + C*!D*!H*!I*L*M + !A*F*!I*L*M + C*!D*H*!I*M +D*E*!I*L*M

V5 = !B*D*E*!F*!G*!H*I*!J*!K*!L*M + !B*E*!F*!G*!H*!I*!J*!K*!L + C*!D*!H*!I*L*M +!D*!I*!J*K*M + !A*!G*!I*J*M + C*!D*H*!I*M + A*!B*!I*J*M + D*E*!I*L*M + !A*F*!I*M + E*!I*!L*M

PLA2
Inputs: A, B, C, D, E, F, G, H, I, J, K
Outputs: Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8

Q1 = !A*!B*!C*!D*E*!G*I*!J + A*D*!E*F*!G*I*!J + A*!B*!E*F*I*!J + C*D*E*F*I*!J + A*C*G*!H + A*!C*!F + A*!C*!D + A*!B*!D +A*E*G + A*!I + A*J

Q2 = A*D*!E*F*!G*I*!J + !A*D*E*!G*I*!J + C*D*E*F*I*!J A*B*!D*F + F*G*!H*I + D*!F*H*I + !A*B*F*J + F*G*!H*J + D*!F*H*J + B*!I*!J + B*!E*!H + B*!C*H + B*C*E + B*E*G + B*D

Q3 = A*D*!E*F*!G*I*!J + !B*C*!D*F + A*C*G*!H + F*G*!H*I + F*G*!H*J + C*!I*!J + !B*C*!E + B*C*E C*F*J + !A*C

Q4 = !A*!B*!C*!D*E*!G*I*!J + !A*B*C*!G*I*!J + B*C*!E*H*I + F*G*!H*I + D

Q5 = A*D*!E*F*!G*I*!J + !A*B*C*!G*I*!J + !A*C*D*!G*I*!J + B*D*F*!G*I*!J + A*B*C*D*F*J + F*G*!H*I + F*G*!H*J + E*F*!I + E*F*H + E*F*J + E*!F

Q6 = !B*!C*D*G*!I*!J*!K + !A*B*D*E*!I*!J*!K + A*B*D*E*G*I*!J + !E*G*!H*!I*!J*!K + !A*C*D*!G*I*!J + B*D*F*!G*I*!J + !D*!H*!I*!J*!K + C*D*E*I*!J + !B*!C*F*!J + C*F*!I*!J + !B*C*!D*F + A*B*!D*F + !A*B*F*J + !B*D*F*J + E*F*!I + C*F*G + E*F*H + E*F*J

Q7 = A*!B*!E*F*I*!J + !A*B*E*I*!J + C*F*G + G*!J + H

Q8 = H

End-User Development Framework for Embedded System Applications

Miroslav Sveda
Brno University of Technology, Faculty of Information Technology
Czech Republic
sveda@fit.vutbr.cz

Abstract

This paper presents principles of an end-user development support for embedded system networking. The presented approach offers a design framework consisting of reusable patterns for Internet-based embedded system applications. The framework provides a development environment kernel that can be adapted for various embedded system application domains. It stems from (1) the IEEE 1451.1 smart transducer interface standard, which is an object-based networking model supporting group messaging, and (2) the Internet Protocol multicast communication, mediating efficient and unified access to distributed components through both wired and wireless intranets. As an example, this paper focuses on refinement of those concepts for smart sensor application development support and on their utilization for a gas pipes pressure measurement system as an application example. The paper brings this scheme in form suitable not only for framework builders, but also for end-user developers.

1. Introduction

The design framework, presented in this paper as a flexible design environment kernel stemming from meta-design conception, is rooted in the IEEE 1451.1 standard specifying smart transducer interface architecture. That standard provides an object-oriented information model targeting software-based, network independent, transducer application environments. The framework enables to unify interconnections of embedded system components through wireless networks and Ethernet-based intranets, which are replacing various special-purpose Field-busses in industrial applications [15].

Two additional technologies, namely publish-subscribe messaging and Internet Protocol (IP) multicasting, which offer scalable and traffic-saving solutions important in the context of contemporary Internet, complement the framework providing design patterns reusable for various networked embedded system applications. The schemes discussed can properly interplay with each other and can deliver suitable support for Internet-based embedded systems design. This paper focuses on utilization of this framework for a gas pipe-line pressure measurement system as a real-world application example, which comprises several groups of smart pressure and temperature sensors that clients can access effectively through Internet. Each sensor group is supported by an active web page that provides clients with transparent and efficient access to pressure measurement services.

The paper discusses in the following section meta-design approach utilizable for creating flexible design environment appropriate for various application domains of networked embedded systems. Next section reviews principles of the proposed design framework aiming at Internet-based embedded systems. Its three subsections introduce subsequently IEEE 1451 package of communication standards, client-server and publish-subscribe communication concepts, and IP multicasting as main components of the design pattern forming the kernel of the generic development environment. While the section 4 briefly introduces concepts of the flexible development environment that enhances the framework for networked smart sensor applications design, the section 5 presents in more detail an example of deployment of this framework for creating networked pressure measurement along gas pipes including some implementation details. The example covers network and node configuration and implementation concepts, including smart sensor implementation.

2. Meta-design for end-user development

Component-based development involves multiple roles [9]. Framework builders create the infrastructure for components to interact; developers identify suitable domains and develop new components for them; application assemblers select domain-specific components and assemble them into applications; and end users employ component-based applications to perform daily tasks. There is room for a fifth role in this pipe-line: end-user developers positioned between application assemblers and end users. These end-user developers are able to tailor applications at runtime because they have both domain expertise and technical know-how. They would interact with applications to adjust individual components, and modify existing assemblies of components to create new functionality. Furthermore, they can play a critical role when compo-

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

nent-based systems have to be redesigned for new requirements. End-user development activities can range from customization to component configuration and programming.

Meta-design characterizes objectives, techniques, and processes for creating new environments allowing end users to act as designers [4]. In all design processes, two basic stages can be differentiated: design time and use time. At design time, system developers create environments and tools. In conventional design they create complete systems. Because the needs, objectives, and situational contexts of users can only be anticipated at design time, users often find the system unfit for their tasks at use time. Thus, they require adaptation of the existing environment and tools for new applications. Meta-design extends the traditional notion of system development to include users in an ongoing process as co-designers, not only at design time but throughout the entire life-cycle of the development process. Rather than presenting users with closed development systems, meta-design provides them with concepts and tools to extend the system to fit their needs. Hence, meta-design promotes designing the design process.

Not surprisingly, meta-design relates to design in the same way as meta-modeling relates to modeling [2, 10]. But, while modeling and meta-modeling are identical activities with the only difference of interpretation, designing and meta-designing are targeted differently. Model, which is the object of modeling, remains an abstract notion in the similar sense as meta-model with the only exception of abstraction level. On the contrary, objects of design and meta-design differ: in the former case the process produces an artifact, in the letter case it is design of a design process including related development environment. By the way, model-based design and development – see the domain related papers [1, 6, 18] – create another glue in between designing and modeling.

This paper discusses a deployment of meta-design principles for building up a flexible design framework focused on embedded systems and their components interconnected by Internet. Necessarily under-designed open source tools and techniques create design spaces for end-user developers. The paper demonstrates both the use of this framework for implementation of development environments aimed at various Internet-based smart sensor applications and, concurrently, the utilization of this framework for development of pressure measurement along gas pipes.

3. Design framework

Main components of the framework, which forms kernel of the generic development environment, encompass IEEE 1451 package of communication standards, client-server and publish-subscribe communication patterns, and IP multicasting.

3.1. IEEE 1451 architecture

The IEEE 1451 package consists of the family of standards for a networked smart transducer interface that include namely (see Figure 1) (i) a smart transducer software architecture, 1451.1 [5], targeting software-based, network independent transducer applications, and (ii) a standard digital interface and communication protocol, IEEE 1451.2, for accessing the transducer or the group of transducers via a microprocessor modeled by the 1451.1 standard. The next three standard proposals extend the original hard-wired parallel interface 1451.2 to serial multi-drop 1451.3, mixed-mode (i.e. both digital and analogue) 1451.4, and wireless 1451.5 interfaces.

The 1451.1 software architecture provides three models of the transducer device environment: (i) the object model of a network capable application processor (NCAP), which is the object-oriented embodiment of a smart networked device; (ii) the data model, which specifies information encoding rules for transmitting information across both local and remote object interfaces; and (iii) the network communication model, which supports client/server and publish/subscribe paradigms for communicating information between NCAPs. The standard defines a network and transducer hardware neutral environment in which a concrete sensor/actuator application can be developed.

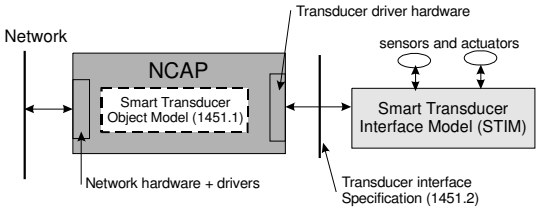


Figure 1: IEEE 1451 configuration example

The object model definition encompasses the set of object classes, attributes, methods, and behaviors that specify a transducer and a network environment to which it may connect. This model uses block and base classes offering patterns for one Physical Block, one or more Transducer Blocks, Function Blocks, and Network Blocks. Each block class may include specific base classes from the model. The base classes include Parameters, Actions, Events, and Files, and provide component classes.

All classes in the model have an abstract or root class from which they are derived. This abstract class includes several attributes and methods that are common to all

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

classes in the model and provide a definition facility for instantiation and deletion of concrete classes including attributes.

Block classes form the major blocks of functionality that can be plugged into an abstract card-cage to create various types of devices. One Physical Block is mandatory as it defines the card-cage and abstracts the hardware and software resources that are used by the device. All other block and base classes can be referenced from the Physical Block.

The Transducer Block abstracts all the capabilities of each transducer that is physically connected to the NCAP I/O system. During the device configuration phase, the description is read from the hardware device what kind of sensors and actuators are connected to the system. The Transducer Block includes an I/O device driver style interface for communication with the hardware. The I/O interface includes methods for reading and writing to the transducer from the application-based Function Block using a standardized interface. The I/O device driver provides both plug-and-play capability and hot-swap feature for transducers.

The Function Block provides a skeletal area in which to place application-specific code. The interface does not specify any restrictions on how an application is developed. In addition to a State variable that all block classes maintain, the Function Block contains several lists of parameters that are typically used to access network-visible data or to make internal data available remotely.

The Network Block abstracts all access to a network employing network-neutral, object-based programming interface supporting both client-server and publisher-subscriber patterns for configuration and data distribution.

The paper demonstrates use of the IEEE 1451.1 architecture as a building block of the framework, which is aimed at Internet-based smart sensor applications, and utilized for development of pressure measurement along gas pipes.

3.2. Communication patterns

The majority of communication protocols provide a client-server style of communication. In case of sensor communications, the client-server pattern covers both configuration of transducers and initialization actions. If the client wants to call some function on server side, it uses a command *execute*. On server side, this request is decoded and used by the function *perform*. That function evaluates the requested function with the given arguments and, after that, it returns the resulting values to the client.

The client-server pattern corresponds to remote procedure call (RPC), which is the remote invocation of operations in a distributed context [3]. To be more precise, the RPC interaction considered in this paper provides a synchronous client-server communication, i.e. the client is waiting for a server's response before completion the RPC actions related to the current call. Evidently, the client-server communication style relates to point-to-point message passing called as unicast.

The subscriber-publisher style of communication, [3], can provide efficient distributions of measured data. All clients, wishing to receive messages from a transducer, register themselves to the group of its subscribers using the function *subscribe*. After that, when this transducer generates a message using the function *publish*, this message is effectively delivered to all members of its subscribing group.

The interaction publish-subscribe relates to point-to-multipoint or multipoint-to-multipoint message passing. While there is a possibility to implement multipoint using multiple point-to-point unicasts, it is much more efficient to utilize elaborate multicast techniques, namely multicast routing. The basic principles of the network layer multicast in the Internet environment are discussed in the following section.

3.3. Multicasting

Traditional network computing paradigm involves communication between two network nodes. However, emerging Internet applications require simultaneous group communication based on multipoint configuration propped e.g. by multicast IP, which saves bandwidth by forcing the network to replicate packets only when necessary. Multicast improves the efficiency of multipoint data distribution by building distribution trees from senders to sets of receivers [8].

The functions that provide the Standard Internet Multicast Service can be separated into host and network components. The interface between these components is provided by IP multicast addressing and Internet Group Management Protocol (IGMP) group membership functions, as well as standard IP packet transmission and reception. The network functions are principally concerned with multicast routing, while host functions can also include higher-layer tasks such as the addition of reliability facilities in a transport-layer protocol.

IP multicasting is the transmission of an IP datagram to a host group, a set of zero or more hosts identified by the single IP destination address of class D. Multicast groups are maintained by IGMP (IETF RFC 1112, RFC 2236). Multicast routing considers multicasting routers equipped with multicast routing protocols such as DVMRP (RFC 1075), MOSPF (RFC 1584), CBT (RFC 2189), PIM-DM (RFC 2117), PIM-SM (RFC 2362), or MBGP (RFC 2283). For Ethernet-based Intranets, the Address Resolution Protocol provides the last-hop routing by mapping class D addresses on multicast Ethernet ad-

dresses. Certainly, the routing protocol architecture must be extended by wireless network routing protocols in case multi-hop sensor networks complement the wired Internet.

The paper demonstrates use of above mentioned communication patterns supported by IP multicast for implementation of communication means aimed at Internet-based smart sensor applications, and describes how to apply them for pressure measurement along gas pipes.

4. Development environment

Development systems have to support important concepts and methods by their tools for complete design and development life cycle of applications belonging to considered application domains. The toolset related to the discussed design framework includes also original tools targeting primarily front-end parts of specification and design, namely formal specifications and rapid prototyping. Those tools and related techniques were discussed in frame of papers that describe the design principles of sensor-based applications in various application domains, such as smart sensor applications [13] and industrial monitoring and measurements [14, 15, 17].

More detailed information about original techniques and tools including supporting principles can be found in the PhD theses, which were prepared in frame of the related research dealing with timed system specifications [7], object-oriented specifications [11], and reverse specifications and modeling [12]. A paper devoted to the flexible development environment and its adaptation for safety and security-critical sensor-based applications is currently under preparation.

5. Design case study

This section demonstrates the above-introduced concepts applied to the design of a gas-pipes pressure measurement system [14, 16]. The developed system comprises several groups of smart pressure and temperature sensors, interconnected by wired intranets and/or wireless networks that clients can access effectively through Internet.

Each sensors group is supported by an active web page with Java applets that, after downloading, provide clients with transparent and efficient access to pressure measurement services over such geographically distributed objects as the considered large systems of gas pipes. The complete system involves several groups of smart pressure sensors complemented by temperature sensors that enable computing of temperature corrections [13] and additional services including self-calibrations. Last paragraph of this section restates development framework refinements through a design pattern provided for this application.

5.1. Network configuration

Each sensor group is supported by an active web page with Java applets that, after downloading, provide clients with transparent and efficient access to pressure measurement services.

In this case, clients communicate to transducers using a messaging protocol defined by client-server and subscriber-publisher patterns employing 1451.1 Network Block functions. A typical configuration includes a set of smart pressure sensors generating pressure values for the users of those values. To register itself for a specified group of sensors, the user — playing the role of either subscriber or client — opens the related server's web page with the relevant Java applet. This applet is, after uploading to the subscriber/client site, started on subscriber/client's computer, which launches communications with a group of transducers allowing Java clients to connect and subscribe to the smart sensors. Java can directly support both client-server and subscriber-publisher application architectures as the core Java specifications include TCP/IP and UDP/IP networking APIs.

The developed Java applet uses the core java.net package to implement both client-server and subscriber-publisher application distribution allowing to access smart sensors and supporting nodes. The applet consists of a series of object classes, including multi-threaded applet environment, animation, and UDP/IP-based subscriber and TCP/IP-based client communications. The subscriber/client software implemented in Java enables applets to be included in a web server HTML page, and run under a regular web browser on subscriber/client side. The subscriber/client communicates with the transducer by standard UDP/TCP sockets employing IP multicast.

The communication scheme applies multicast both for distributing measured values from a transducer to a group of subscribers/clients registered by the web server for this transducer, and for spreading commands of a client to a group of transducers registered for this client.

5.2. System implementation concepts

In the transducer's 1451.1 object model, basic Network Block functions initialize and cover communication between a client and the transducer, which are identified by unique unicast IP addresses. The client-server style communication, which in this application covers both the configurations of transducers and initialization actions, is provided by two basic Network Block functions: *execute* and *perform*.

The standard defines a unique ID for every function and data item of each class. If the client wants to call some

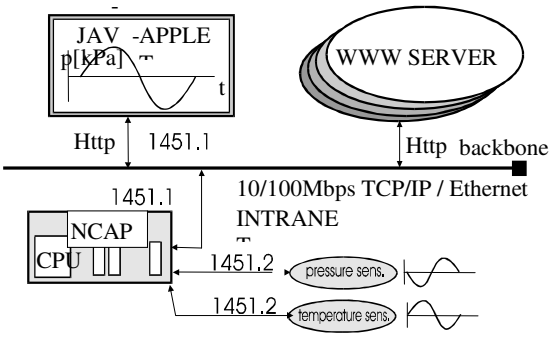


Figure 2: Sensor node configuration example

function on server side, it uses command *execute* with the following parameters: ID of requested function, enumerated arguments, and requested variables. On server side, this request is decoded and used by the function *perform*. That function evaluates the requested function with the given arguments and, in addition, it returns the resulting values to the client. Those data are delivered by requested variables in *execute* arguments.

The subscriber-publisher style of communication, which in this application covers primarily distribution of measured data, but also distribution of group configuration commands, employs IP multicasting. All clients wishing to receive messages from a transducer, which is joined with an IP multicast address of class D, register themselves to this group using IGMP. After that, when this transducer generates a message by Block function *publish*, this message is effectively delivered to all members of this class D group, without unnecessary replications and repeated transmissions.

The Network Block abstracts all access to a network employing network-neutral, object-based programming interface. The network model provides an application interaction mechanism supporting both client-server and publisher-subscriber paradigms for event and message generation and distribution.

5.3. Node configuration

The primary communication scheme, which is based on publish-subscribe pattern, applies multicast both for distributing measured values from a transducer to a group of clients registered by the www server for this transducer, and for spreading commands of a client to a group of transducers registered for this client. Those commands can specify e.g. individual subgroup's sampling frequencies and/or events for launching irregular publishing such as a limit value crossing.

A typical node, depicted on Figures 2 and 3, consists of STIM (Smart Transducer Interface Module) connected with PSD sensor for pressure measurements, and with auxiliary temperature sensor for signal conditioning. Of course, NCAP can be either embedded in a complex smart sensor, or shared among more simple smart sensors. On the other hand, from the viewpoint of Internet, only NCAP is directly addressable being equipped by its own IP address. Therefore, we can also denote as smart sensor the device consisting of an NCAP accessing one or more STIMs with connected sensors.

To register itself for a specified group of sensors, the client opens a related server's web page with the relevant Java applet. This applet is, after uploading to the client site, started on client's computer, what launches communication with the dedicated group of transducers.

5.4. Smart sensor implementation

This subsection discusses, as an example, the pressure sensors with reflected laser beam and diffractive lens. The sensitive pressure sensor is based on a nitride membrane and an optoelectronic read-out subsystem. Measured pressure values are transformed into related thick-layer nitride membrane deflections. The nitride membrane serves as a mirror for laser beam, and it can move the related reflected laser mark. The mark's position is sensed using position-sensing device, which is a fotolateral diode. Diode double current signal is amplified and conditioned digitally by the ADuC812 microcontroller. This microcontroller provides also the IEEE 1451.2 interface.

The sensing subsystem combines two principles that provide both high precision and wide range pressure measurements. Large displacements are measured by the position of reflected focused laser beam. Small position changes are measured by one-side layer diffractive lens principle. Sensor output signal is conditioned in digital by the ADuC812 single-chip microcontroller, which provides

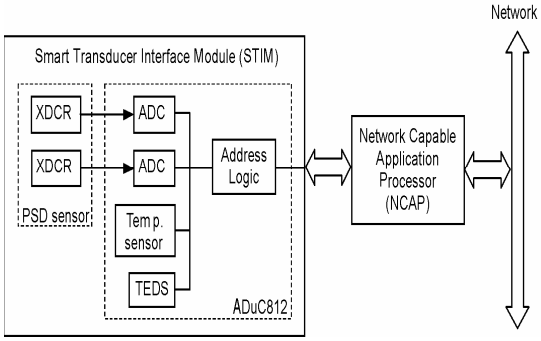


Figure 3: Sensor node implementation example

the IEEE1451.2 interface as one of its communication ports. This microcontroller calculates the position of the light spot and converts that position on the measured pressure using an internal table. Figure 3 depicts principles of

the implementation of that smart sensor. The STIM contains (1) a PSD sensor with two analog differential transducers (XDCR), (2) a microcontroller ADuC812 with nonvolatile memory containing a TEDS field (Transducer Electronic Data Sheet) that props IEEE 1451.2 storing sensor specifications, (3) a TII (Transducer Independent Interface), (4) a temperature sensor necessary for signal conditioning, (5) an analogue-to-digital conversion units (ADC), and (6) a logic circuitry to facilitate communication between STIM and NCAP.

The ADuC812 microcontroller, the basic building block of the smart pressure sensor electronics, includes on-chip high performance multiplexers, ADCs, DACs, FLASH program and data storage memory, an industrial standard 8052 microcontroller core, and supports several standard serial ports. The microcontroller may also utilize nonvolatile memory containing a TEDS field and ten-wire TII that prop IEEE 1451.2.

5.4. Design pattern

In conclusion of this case study it could be helpful to restate the key refinements of the proposed framework through building up a design pattern based on its basic abstract components, i.e. IEEE 1451 architecture, communication procedures and IP multicasting, which introduce a more detailed structure reusable for concrete applications.

The 1451.1 object model provides skeleton supporting individual components. Its Network Block is refined so that it enables to access data-link layer communication services through unicast on IP with client-server procedure for start-up configuration and run-time maintenance, or through IP multicast with publish-subscribe procedure for run-time process measurements on both application data users and transducers sides. This refinement covers also selection of the most appropriate multicast routing protocol for local Internet traffic in case when the relevant parts of the network are accessible through routers.

The Transducer Block includes methods for reading and writing to transducers from the application-based Function Block using the standardized interfaces. The I/O device driver provides both plug-and-play capability and hot-swap feature for each transducer. It enables run-time reconfiguration of sensors that can support robustness of the system or improve measurement efficiency.

The Function Block contains measurement application code. In the current case it prescribes sampling times, data filtering, linearization, conversions and transformations improving measurement accuracy and stability. The Function Block contains lists of parameters used to access network-visible data and, concurrently, to make internal data available remotely. The Function Block enables in this case to compute pressure profiles along the pipeline, pressure or temperature gradients, and the speed of pressure or temperature changes in time.

The measurement system arrangement along a pipeline is depicted on the Figure 4.

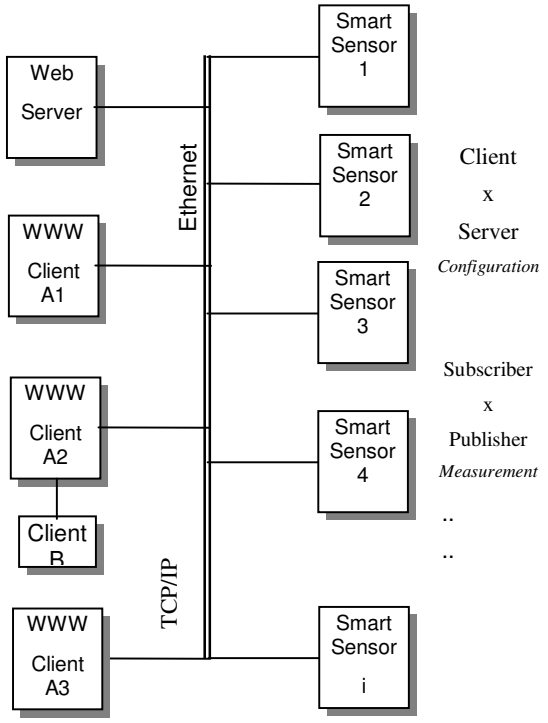


Figure 4. The measurement system network arrangement

This figure shows active nodes participating in data exchange on an intranet in frame of client (Client A1…A3) – server (Web server) configuration for client registration, and publisher (Smart Sensor 1…3) – subscribers (a subset of Clients A1…A3 and B) configuration for measured data transmissions.

6. Conclusions

This paper presents principles of the end-user development support for embedded system networking in form of the developed design framework. The approach supported offers a reusable pattern for Internet-based embedded system applications. It stems from (1) the IEEE 1451.1 smart transducer interface standard, which is an object-based networking model supporting group messaging, and from (2) the Internet Protocol multicast communication, mediating efficient and unified access to distributed components through both wired and wireless intranets. As an example, this paper discusses adaptations and refinement of this framework for smart sensor application

development support and its utilization for a gas pipes pressure measurement system. The paper brings this approach in manner suitable not only for framework builders, but also for end-user developers.

From more general viewpoint, the paper discusses deployment of meta-design principles for creating flexible design environments. Necessarily under-designed open source tools and techniques create design spaces for end-user developers in various application domains of networked embedded systems.

Acknowledgements

The research has been supported by the Czech Ministry of Education in frame of the Research Intention MSM 0021630528: Security-Oriented Research in Information Technology, and by the Grant Agency of the Czech Republic through the grants GACR 102/05/0723: A Framework for Formal Specifications and Prototyping of Information System's Network Applications and GACR 102/05/0467: Architectures of Embedded Systems Networks. The author appreciates contributions of his colleagues Radimir Vrba, Ondrej Rysavy, Frantisek Scuglik, and Petr Matousek from the Brno University of Technology to this work.

References

[1] Balasubramanian, K., et al., Developing Applications Using Model-Driven Design Environments, *IEEE Computer*, pp.33-40, February 2006.

[2] Childs, A., et al., CALM and Cadena: Metamodeling for Component-Based Product-Line Development, *IEEE Computer*, pp.42-50, February 2006.

[3] Eugster, P.T., et al., The Many Faces of Publish/Subscribe. *ACM Computing Surveys*, Vol. 35, pp.114-131, 2003.

[4] Fischer, G. et. al., Meta-Design: A Manifesto for End-User Development. *Communications of the ACM*, Vol.47, No.9, pp.33-37, 2004.

[5] *IEEE 1451.1, Standard for a Smart Transducer Interface for Sensors and Actuators -- Network Capable Application Processor (NCAP) Information Model*. IEEE, Washington, D.C., 2000.

[6] Jackson, E.K., and J. Sztipanovits, Correct-ed through Construction: A Model-based Approach to Embedded Systems Reality. *Proceedings 13th Engineering of Computer-Based Systems*, IEEE Computer Society, Los Alamitos, CA, pp.164-173, 2006.

[7] Matousek, P., *Symbolic Data Structures for Parametric Verification*, PhD Dissertation, Brno University of Technology, Brno, Czech Republic, 2005.

[8] Miller, C.K., Multicast Networking and Applications, Addison-Wesley, Reading, Massachusetts, 1999.

[9] Morch, A.I., et al., Component-Based Technologies for End-User Development. *Communications of the ACM*, Vol.47, No.9, pp.59-62, 2004.

[10] Nordstrom, G., et al., Metamodeling – Rapid Design and Evolution of Domain-Specific Modeling Environment, *1999 Proceedings Engineering of Computer-Based Systems*, IEEE Computer Society, Los Alamitos, CA, pp.177-186, 1999.

[11] Rysavy, O., *Specifying and Reasoning in the Calculus of Objects*, PhD Dissertation, Brno University of Technology, Brno, Czech Republic, 2005.

[12] Scuglik, F., *Tools for Communication Protocols Evaluation*, PhD Dissertation, Brno University of Technology, Brno, Czech Republic, 2007.

[13] Sveda, M. and R. Vrba, An Integrated Framework for Internet-Based Applications of Smart Sensors. *IEEE Sensors Journal*, Vol.3, No. 5, pp.579-586, 2003.

[14] Sveda, M., A Design Framework for Internet-Based Embedded Distributed Systems. *Proceedings of the International IEEE Conference and Workshop ECBS'2004*, IEEE Computer Society, Los Alamitos, CA, pp.113-120, 2004.

[15] Sveda, M., et al., Introduction to Industrial Sensor Networking. A book chapter in: Ilyas, M., and I. Mahgoub, (Eds.), 2005. *Handbook of Sensor Networks: Compact Wireless and Wired Sensing Systems*, CRC Press LLC, Boca Raton, FL, 2005.

[16] Sveda, M. and R. Vrba, Internet-Based Embedded System Architectures. *Proceedings of the International Joint Conference on e-Business and Telecommunications ICETE/ICE-B 2006*, INSTICC and IEEE, Setúbal, Portugal, pp. 63-68, 2006.

[17] Sveda, M. and R. Vrba, An Approach to the Design of Networked Embedded Systems. *Proceedings of the 16th IFAC World Congress*, IFAC, Prague, Czech Republic, pp.2223-2228, 2005.

[18] Szemethy, T., et al., Model Transformations in the Model-Based Development of Real-time Systems. *Proceedings 13th Engineering of Computer-Based Systems*, IEEE Computer Society, Los Alamitos, CA, pp.177-186, 2006.

Agile Development Methodology for Embedded Systems: A Platform-Based Design Approach

Lucas Cordeiro[1,3], Raimundo Barreto[1], Rafael Barcelos[3], Meuse Oliveira[2], Vicente Lucena[1], and Paulo Maciel[2]

[1]Departamento de Ciência da Computação - Universidade Federal do Amazonas (UFAM), Brazil
rbarreto@dcc.ufam.edu.br and vicente@ufam.edu.br

[2]Centro de Informática - Universidade Federal do Pernambuco (UFPE), Brazil
$\{mnoj, prmm\}$@cin.ufpe.br

[3]BenQ Mobile Phones - Research and Development Center, Brazil
$\{lucas.cordeiro, rafael.barcelos\}$@benq.com

Abstract

This paper describes an agile development methodology which combines agile principles with organizational patterns and adapts them to build embedded real-time systems focusing on the system's constraints. The hardware/software partitioning and platform-based design are used in the proposed methodology to support the embedded system designer meet the system's constraints in an iterative and incremental way and to reduce substantially the design time and cost of the product. To discuss the strengths and weakness of this methodology, a case study involving a pulse oximeter is also presented.

1 Introduction

The micro-controllers becoming cheaper, smaller and more reliable make it economically attractive to be used as computer systems in several appliances. Approximately 3 billion of micro-controllers (μC) are sold each year and smaller μC (4-,8-, and 16-bit) are dominating the market and adding value to products [6]. The embedded computer systems are used in a wide range of system from machine condition monitoring to airbag control systems. As the system complexity increases, its development lifecycle is also affected. Because of that, system development methodologies must be applied in order to manage the team size, to manage the product requirement *(scope)* and to meet the project's constraints *(time-to-market and costs)*.

Nevertheless, many development methodologies that are used to produce software that runs on the personal computers (PC's) are not appropriate for developing embedded real-time systems. This kind of system contains very different characteristics such as dedicated hardware and software, and constraints that are not commum to PC's based systems (e.g. energy consumption, execution time, memory footprint). Moreover, many embedded systems engineers do not have good software engineering skills. They have hardware development skills and often use programming languages to solve the problems at hand in an empirical way [5]. Another important point is that some classes of embedded real-time systems may put lives or business-critical functions at risk *(mission criticality)*. Therefore, these systems should be treated differently from the case where the only cost of failure is the project's investment.

Based on this context, we propose a development methodology based on the agile principles such as *adaptive planning, flexibility, iterative and incremental approach* in order to make easier the development of embedded real-time systems. To achieve that, this methodology is composed by best practices from Software Engineering and Agile methods (Scrum and XP) which aim at minimizing the main problems present on the embedded software development context *(i.e. requirement volatility and risk management)*, and by others practices that are needed to achieve embedded real-time systems *(i.e. platform-based design* [13]). On this paper, this methodology and its components (roles, process and tools) are described.

The remainder of this paper is organized as follows: Section 2 summarizes the related works. Section 3 overviews the agile methods and patterns that were integrated into the proposed methodology. Section 4 describes the proposed agile methodology and section 5 shows the application of

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

the proposed methodology in the development of the pulse oximeter. Finally, section 6 summarizes this paper and identifies the next steps from this research.

2 Related Works

Embedded software development teams usually do not make use of development methodologies or any other more complex software engineering concept [5]. There are different reasons that explain this fact, but the main one is the developers' lack of maturity related to software engineering practices. Nevertheless, we identified in the literature through a bibliographical review, three different development methodologies that allowed us to evaluate the state of the art in this context and to support us during the definition of our proposed approach.

One of the results from this review is a paper that describes the experience of applying Agile approaches to the development of firmware for the Intel Itanium processor family [5]. In this paper, Greene identified the agile practices that his team successfully applied, but he did not take into account the hardware related development, one of the main parts of this kind of development. Greene only mentioned that another Intel team was applying agile concepts and they were having good results. Even so, the comments retrieved from this paper regarding the application of agile concepts were very useful during the definition of our approach since it supports the benefits of using agile concepts in contexts beyond object oriented software development.

The second one is the methodology proposed by Gajski [4] which aims to develop embedded systems by formally describing the system's functionalities in an executable language rather than a natural language. The executable specification is refined through the system-design tasks of allocation, partition, and refinement. Estimators are also used in order to explore design alternatives. Since the system components are defined formally then components are implemented by just compiling the component's functional description into machine code. This methodology has already been applied to several embedded systems projects and has influenced our proposed methodology. However, this methodology assumes that all requirements are captured before applying the partitioning algorithms.

Finally, Manhart and Schneider [8] related a successful industrial experience when partially adopting agile methods in the production of software for embedded systems. Indeed they made slight modifications in a well established software development process for the automotive branch adopting some agile elements in order to adequate their process to new needs as flexibility and high speed software production. As pointed out in the paper many other application areas may benefit from their experiments, nevertheless the authors did not presented any measurement results that could prove their expectations.

The difference of the proposed methodology compared with other methodologies can be described as follows: *(i)* our methodology aims to tradeoff flexibility and performance by adopting highly programmable platforms, *(ii)* hardware/software estimation and partitioning techniques are used in order to explore design alternatives and meet the system's constraints, *(iii)* by making use of the iterative and incremental approach, the product development can be broken in a sequence of iterations and implemented in an incremental way, *(iv)* as the system functionalities increases iteration by iteration then the proposed methodology offers clearly an iterative process where the designer can validate the partition of a system specification produced by algorithms, *(v)* and last but not least, the proposed methodology adopts an adaptive planning which makes it possible to embraces changes even late in the development process.

3 A Brief Look at the Agile Methods and Patterns

In this section, a brief look at the agile principles, methods, and patterns that were used in the proposed methodology is presented. It identifies the main product development and management practices of the XP and Scrum methods respectively.

3.1 Extreme Programming

The most recognizable agile method is eXtreme Programming (XP) which is very communication-oriented and team-oriented [1]. XP is composed of 12 core practices and some of its main characteristics that were integrated into the proposed methodology include: Refactoring practice *(i)* which is the process of changing a software system in such a way that it does not alter the external behavior of the code and at the same time improves its internal structure.

In the Continuous Integration practice *(ii)*, the code is compiled and tested in an automated process every time it is checked-in. Test-driven development practice *(iii)* means that the unit tests are written by the developers before coding. These unit tests are automated tests that test the functionality of pieces of the code. In the Coding Standard practice *(iv)*, everyone involved in the project needs to follow the same code style. It specifies a consistent format for source code, within the chosen programming language.

XP promotes an evolutionary approach to design the system by using the first three practices described above. The main benefit of this approach is that the system grows in an incremental way and it aims to reduce project's risk and uncertainty too early *(risk management)*. Section 4 describes how these XP practices were adapted into the proposed methodology.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

3.2 Scrum

Scrum is a simple and straightforward approach to manage the software development process based on the assumption that environmental (i.e. people) and technical (i.e. technologies) variables are likely to change during the process [12]. Scrum is composed of 14 practices and some of its main characteristics that were integrated into the proposed methodology include: Sprint practice *(i)* is the iteration organized in 30-calendar-day. The Sprint Planning practice *(ii)* consists of two meetings.

In the first meeting, the product backlog which contains a list of features, use cases, enhancements, and defects of the system is refined and re-prioritized by the product owner, stakeholders and goals for the next iteration are chosen. In the second meeting, the Scrum team figures out how to achieve the requests and creates the sprint backlog that contains detailed tasks to be accomplished in the current iteration. In the Sprint Review practice *(iii)*, the Scrum team presents the results obtained at the end of each iteration by showing the working software for the product owner, customers and other stakeholders. In the Daily Scrum practice *(iv)*, daily meetings are held at the same place and time with special questions to be answered by the Scrum team.

Scrum employs the empirical process control model, i.e. the practices aim to inspects the condition of activities and empirically determines what to do next in order to produce the expected outcomes *(product)*. The productivity and quality strongly depend on both skills and motivation of the people involved in the process. Section 4 shows how the Scrum practices were adapted into the proposed methodology.

3.3 Patterns for Agile Software Development

The agile patterns described by [3] can be combined with XP and Scrum agile methods with the purpose of structuring the software development process of the organizations. These patterns are split into four different pattern languages as follows: The *project management pattern language* provides a set of patterns that help the organization manage the product development, clarify the product requirements, coordinate project's activities, generate system's build, and keep the team focus on the project's primary goals.

The *piecemeal growth pattern language* provides a set of patterns that help the organization define the high-level management and amount of team members per project, ensure and maintain customer satisfaction, communicate the system requirements, and ensure a common vision for all people involved in the product development team. The *organizational style pattern language* provides a set of patterns that help the organization eliminate project's overhead and latency, ensure that the organization structure is compatible with the product architecture, organize work to develop products by geographically distributed teams, and ensure that the market needs will be met.

The *people and code pattern language* provides a set of patterns that help the organization define and keep the architecture style of the product, ensure that the architect is materially involved in implementation, and assign feature development to people in nontrivial projects. The *software configuration management pattern language* is not part of the organizational patterns but they were integrated into the proposed methodology. These patterns were defined by [2] and they offer patterns that help the development team define mechanisms for managing different versions of the work products, develop code in parallel with other developers and join up with the current state of development line, and identify what versions of code make up a particular component.

4 Proposed Agile Development Methodology

The proposed methodology aims to define roles and responsibilities and provide processes, lifecycle, practices and tools to be applied in embedded real-time system projects. It contains three different processes groups that should be used during the system development: *system platform, product development and management*.

The system platform processes group aims to instantiate the platform for a given product. It means that the system designer must choose the system components that will be part of the architecture and API platforms from a platform library. After that, the system designer has still the possibility to customize the architecture and API platforms in order to meet the application constraints. The customization process is carried out by programming the designer-configurable processors and runtime-reconfigurable logic integrated into the platform. The customization process is carried out by successive refinements in an iterative and incremental way into the proposed methodology.

The product development processes group offers practices to develop the application's components and integrating them into the platform. The functionalities which make up the product are partitioned into either hardware or software elements of the platform. The partitioning algorithms used to carry out this task take into account the energy consumption, execution time, and memory size of the application's components. The mechanical design is also part of this processes group, but it is out of the scope of this paper. The partitioning technique is also applied in an iterative and incremental way.

The product scope, time, quality, and costs parameters are monitored and controlled by the product management processes group. These parameters also influence the sys-

tem platform and product development processes groups. When the project starts with an infeasible project plan which needs corrective actions to be carried out then this processes group aims to get the project back on the track and ensure that the project's parameters are met. The product management processes group consists of the practices promoted by the Scrum agile method as well as the agile patterns described in Section 3. The next subsections are concerned with describing the processes groups, roles and responsibilities, and the processes lifecycle of the proposed methodology.

4.1 System Platform Processes Group

The system platform processes group is composed of the following processes: *product requirements, system platform, product line, and system optimization*. The *product requirements process* aims to obtain the system's requirements (functional and non-functional) that are relevant to determine the system platform in which the product will be built. The *platform instance process* helps the development team define the system platform by making use of a set of design tools and benchmarks.

After defining the system platform, the *product line process* helps the development team setup the repository in which the system platform components will be available to the product development. This process also allows the development team to implement and integrate system's functionalities into the system and release new product versions into the market. After implementing and integrating the system's functionalities into the product development line, the *system optimization process* provides activities to ensure that system's variables such as execution time, energy consumption, program size and data memory size satisfy the application constraints.

4.2 Product Development Processes Group

The product development processes group is composed of the following processes: *functionality implementation, task integration, system refactoring, and system optimization*. The *functionality implementation process* ensures that test cases are created for every product's functionality. This process helps increase the product quality and reduce the creation of complex functions. The *task integration process* provides means to integrate new implemented functionalities into the development line of the product without forcing the other team members to work around it.

The *system refactoring process* helps the development team identifies opportunity to improve the code and changing it without altering its external behavior. After refactoring the code, the *system optimization process* allows the development team to optimize small part of the code by making use of profiler tools that monitor the program and tells where, for instance, it is consuming time, energy, and memory space. This process guarantees that software metrics meets the system constraints.

4.3 Product Management Processes Group

The product management processes group is composed of the following processes: *product requirements, project management, bug tracking, sprint requirements, product line, and implementation priority*. The *product requirements process* (that also belongs to the system platform processes group) aims to obtain the system's requirements (functional and non-functional) that must be part of the product. The *project management process* allows the development team to implement the system's requirements by managing the product and sprint backlog, coordinating activities, generating system's build, and tracking the product's bug.

The *bug tracking process* allows the product leader to manage the lifecycle of the project's issues (bug, task, and enhancement) and provide the needed information about the product quality through the release notes for the end user. The *sprint requirements process* allows the development team to analyze, evaluate, and estimate the system's functionalities before starting a new project's sprint. This information is included into the sprint backlog which will help the development team partition the system functionalities into either hardware or software before starting the sprint.

The *product line process* guarantees that the system functionalities implemented during the sprint will be integrated into the product development line. This process also helps the development team to release new product versions into the market. The *implementation priority process* helps the product leader manage any kind of interruptions that may impact the project's goals. This process guarantees that the project's tasks are 100 percent completed after initiated.

4.4 Roles and Responsibilities

The proposed methodology involves four different roles and the responsibility of each role is described as follows:

Platform Owner: Platform owner is the person who is officially responsible for the products that derive from a given platform. This person is responsible for defining quality, schedule and costs targets of the product. He/she must also create and prioritize the product backlog, choose the goals for the sprints, and review the product with the stakeholders.

Product Leader: Product leader is responsible for the implementation, integration and test of the product ensuring that quality, schedule, and cost targets defined by the platform owner are met. He/she is also responsible for mediating between management and development team as well as listening to progress and removes block points.

Feature Leader: Feature leader is responsible for managing, controlling and coordinating subsystem projects, pre-integration projects, external suppliers that contribute to a defined set of features. The feature leader also tracks the progress and status of the feature development (deliverables, integration and test status, defects, and change requests) and reports the status to the product leader.

Development Team: The development team which may consist of programmers, architects, and testers are responsible for working on the product development. They have the authority to make any decisions, do whatever is necessary to do (according to the project's guidelines), and ask for any block points to be removed.

If the product to be developed is small, i.e. it is composed of few components and does not require other development teams to implement the product's functionalities then one product leader and the development team are enough for the product development. On the other hand, if the product is composed by several components and requires other development teams to implement the product's functionalities then the Feature Leader role must be involved in the processes. In this context, one product leader requires feature leaders to manage, control and coordinate components' projects. Therefore, for medium and larger projects, one product leader and several feature leaders and development teams may be involved in the processes.

4.5 Processes Lifecycle

The proposed agile methodology consists of five phases: *Exploration, Planning, Development, Release, and Maintenance*. In the *Exploration phase*, the customers provide requirements for the first product release. These requirements are included into the product backlog by the platform owner. After that, the platform owner and product leader estimate the requirements with no item larger than 3 person-days of effort. In this phase, the development team identifies the application constraints and estimates the system's metrics based on the product backlog items. With this information at hand, the development team is able to define the system platform that will be used to develop the product in the next phases.

In the *Planning phase*, the platform owner and customers identify more requirements and prioritize the product backlog. After that, the development team spends one day to estimate the sprint backlog items and decompose them into tasks. The tasks that make up the sprint backlog must take from 1 to 16 hours to be completed. Explanatory design and prototypes may also be developed at this phase in order to help clarify the system's requirements.

In the *development phase*, the team members implement new functionalities and enhance the system based on the items of the sprint backlog. The daily meetings are held at the same time and place with the purpose of monitoring and adapting the activities to produce the desired outcomes. At the end of the each iteration, unit and functional tests are executed in a continuous integration build. System optimization also takes place during this phase. The last sprint provides the product to be deployed in the operational environment.

In the *Release phase*, the product is installed and put into practical use. During this phase, it usually involves the identification of errors and enhancement in the system services. Therefore, the platform owner and customers decide if these changes will be included in the current or subsequent release. This phase aims to deliver the product and the needed documentation to the customer. The *Maintenance phase* may also require more sprints in order to implement new features, enhancement and bug fixes raised in the release phase.

5 Applying the Proposed Methodology

This section is concerned with describing the application of the proposed methodology in the development of the pulse oximeter equipment. We chose the pulse oximeter as a case study because it was already developed in another work by our research group using an ad hoc development methodology [10]. Therefore, we describe the pulse oximeter in this section only as an example of the application of the proposed methodology in the domain of embedded systems. Generally speaking, the pulse oximeter is responsible for measuring the oxygen saturation in the blood system using a non-invasive method. The architecture of this equipment is shown in Figure 1.

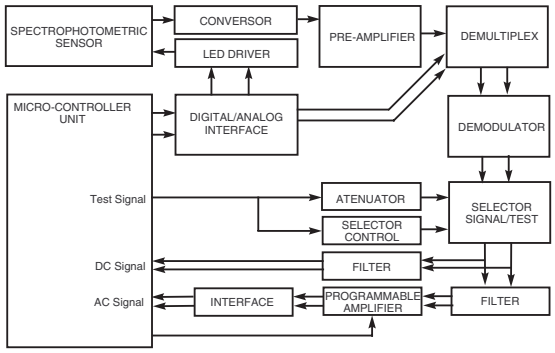


Figure 1. Pulse Oxiemeter Architecture.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

The micro-controller controls the synchronization and amplitude of the led driver, which dispatches non-simultaneous stream pulses to the infrared and red leds. Both leds generate, respectively, infrared and red radiation pulses that cross the finger of a patient. After crossing the finger, a photo-diode catches the radiations level. A sequence of operations occurs until data reaches the micro-controller. Lastly, the micro-controller performs the calculation related to oxygen saturation level based on data received, and shows the result on a display. The final product contains about 5000 lines of C code. Due to timing and energy constraints, the sensor signals excitation and conditioning were implemented using hardware components while the control algorithm and the signal conversion system were implemented in software by the micro-controller.

The main system's characteristics are depicted as follows: *(i)* the level of the oxygen saturation and cardiac frequency must be shown every second, *(ii)* The user must be able to change the alarm configuration, *(iii)* the user interface of the pulse oximeter equipment must have a keyboard and a graphical display, *(iv)* the design of the system should be highly optimized for life-cycle cost and effectiveness, *(v)* the amount of software defects should be as low as possible, and *(vi)* the power dissipation of the final system should be about 2,35 Watts.

If we develop this product using the proposed methodology then there would be about three people involved, two developers and one product leader. The product would take approximately four sprints of three weeks to be developed, tested and delivered (rough estimation). The next subsections describe only the processes of the proposed methodology that focus on achieving the aims of the pulse oximeter equipment.

5.1 Process for Managing the Product Requirements

This process would help us identify the market needs for the pulse oximeter product line and manage the product requirements. At the beginning of the project, we could arrange a brainstorming meeting in order to capture high-level requirements of the product. After that, we could create an initial product backlog with the purpose of capturing more requirements and creating a first product prototype. The first project iteration would allow us to answer questions such as whether the technology needed for the system exists, how difficult it would be, and whether the engineers would have enough experience using that technology.

We could put much emphasis on delivering the system's functionalities *(i)*, *(ii)*, and *(iii)* in the beginning of the sprints. Delivering these functionalities with highest business value (the business value could range from 1 lowest to 5 highest), could help our customer (e.g., a representative of a hospital) and the product leader get feedback on functionality earlier and allow them to spot any misunderstanding more quickly. At the end of each iteration, the product leader and customer could verify if the product was still feasible or not. If the project was not feasible then it could have been canceled just after the end of the iteration *(risk management)*.

If we develop a new product in which the requirements cannot be expected to be fully available earlier in the development then this process would help us identify more requirements and update the product backlog as the project evolves. We could start the project with a set of functional and non-functional requirements. After running some project's sprint, our understanding about the product would increase and we could identify the requirements that were not captured at the beginning of the project.

5.2 Process for Managing the Project

This process would help us refine and prioritize the product backlog that contains the system's functionalities. In the sprint planning, the product leader and our customer could choose the goals of the next sprint based on the highest business value and risks of the product backlog items. After that, we could have a meeting to consider how to achieve the sprint's goals and to create the sprint backlog. The sprint backlog should contain only tasks in the 4-16 hour range.

During the system development, the sprint backlog could be updated on a daily basis as the activities were being accomplished. The product leader could hold daily meetings at the same place and time with the team members with the purpose of monitoring and controlling the complexity of the tasks. These daily meetings would provide a great feedback to the product leader and create the habit of sharing the knowledge. After starting the sprint, we could implement first the functional requirements and then focus on the non-functional requirements of the system. This approach would help us obtain better optimization results because we would be trying to optimize the global system instead of only parts of the system which sometimes could not lead to the global optimization.

During this phase, system's builds could also be generated on a daily basis which could help our customer clarify the requirements and asses the risks earlier in the development process. At the end of the sprint, the product leader and development team could show the results of the work to the customers. This meeting aims to present the product increment, technology and business situation. These artifacts would help the product leader and customers decide the goals of the next sprint. In addition, after each sprint review there would be a retrospective meeting which has the purpose of collecting the best practices used in the sprint and identifying what could be improved for the next sprint.

5.3 Process for Instantiating the Platform

This process would help us estimate the pulse oximeter metrics in order to define the system platform. To obtain the execution time and energy consumption metrics, we could specify the system's functionalities in the Unified Modeling Language (UML) by creating the class, collaboration and sequence diagrams. UML 2.0 profile could also be used to specify the system functionalities [7], but at the time we wrote this paper there was no tool available to convert the UML 2.0 diagrams into a programming language. The main benefits of the UML 2.0 is that it uses special stereotypes and design rules for modeling the behavior of the application and platform components and it also enables their parameterization.

The CASE (Computer Aided Software Engineering) tools like Together and Rational Rose could be used for the entry of the system model. After specifying the system model in UML using these tools, the code could be generated automatically in the language selected by the system designer (e.g., SystemC, Java, and C/C++). Nguyen et. al. [9] provides a tool that enables the system designer to specify the system model in UML and automatically convert it into SystemC code [9]. After generating the code in the selected language, hardware/software estimation tools could be used to estimate the execution time and energy consumption. We could use the estimation tool developed by our research group that is capable of estimating the execution time and energy consumption based on Assembly code [11].

Therefore, after estimating the system's metrics, we could provide this information to our hardware/software partitioning tool that is being developed as an Eclipse plug-in by our research group. This tool looks for the best partitioning that meets the design constraints. Since most design decisions are driven by constraints then we should incorporate the application constraints into our objective function so that partitions that met constraints would be considered better than those that did not meet. Finally, after instantiating the platform based on the application constraints then we could start developing the product.

5.4 Process for Implementing New Systems Functionalities

This process would help us implement the system's tasks of the pulse oximeter in a systematic way. According to the business value of the system's functionalities defined in the *process for managing product requirements*, we could start implementing the tasks responsible for measuring the oxygen saturation level in the patient's blood. In order to implement this functionality, we should first write the unit test for this functionality and thereafter we should successfully compile the unit test before really writing the functionality's code. If there would be compilation problems then the unit test for this functionality should be fixed.

After successfully compiling the unit test, we could start coding the functionality by following product's coding standard defined at the beginning of the project. The functionality's code would be considered completely implemented only after the team member successfully runs the unit test that was created for the functionality. Therefore, it would ensure that the functionality would be in compliance with its specification and the calculation of the oxygen saturation level in the patient's blood would be correctly performed. If there would be some need for splitting this functionality into different tasks then there would be the need for creating the unit tests for each code piece of the functionality.

5.5 Process for Refactoring the Code

After implementing the system's functionalities, we could identify in further sprints opportunities to improve an existing code. For instance, we could identify during the pulse oximeter project that the level of the oxygen saturation and cardiac frequency functionalities have some tasks in common. Both functionalities need to collect data from the sensor and identify the maximum and minimum signal pulse. Therefore, the application of this process would lead to elimination of duplicated code, reduction of the amount of system's functions, and improve the system performance.

But before improving the code for those tasks in common, we should first create branches in the system repository for not breaking an existing working code. After that, we should verify if there is some need for updating the unit test of the functionality. If there is no need to update the unit tests then we could start improving the code without altering its external behavior. After refactoring the code, we could run the unit test in order to verify if the changes are working correctly. If there is no compilation problem and the unit test dos not fail then we could integrate our changes into the product development line.

After integrating the code, the regression tests could be run in order to check if there is no compilation and semantic problems. If there is no problem then the refactoring would be completed. Another important point when applying the refactoring process is that system functions might also be moved from one system component to another, i.e., the system designers might decide to move a function from software running on one of the processors to a hardware block. For instance, the signal conditioning of the pulse oximeter could be moved from software component to a full-custom logic, an application-specific integrated circuit (ASIC), or reconfigurable logic component. This process would also lead to improve the system performance.

5.6 Process for Optimizing the System

This process would help us identify system's variable that could be optimized in order to meet the system's constraint. To optimize the system's variables, we should first establish the metrics and ensure that the refactoring process has already been applied. After that, we could run our profiler tool [11] to monitor the program and tell where it would be consuming time and energy. In this way, we could be able to find small parts of the program where these system's variables could be optimized. Thereafter, we could optimize the variables under attention by hand. As in refactoring, we could also carry out the changes in small steps. After each step, we should compile, test and run the profiler tool again. If the variable has not been optimized then we should return the changes in the version control system and continue the optimization process until the variables could satisfy the constraints.

6 Conclusion

This paper described an agile development methodology and its application in the development of the pulse oximeter. In order to create the methodology, we chose two agile methods XP and Scrum as well as organizational patterns named in this paper as agile patterns. Scrum is explicitly intended for the purpose of managing agile software development projects. On the other hand, XP is a collection of well-known development practices. The agile patterns provide means to structure the software development process of organizations.

When XP, Scrum and agile patterns are combined they cover many areas of the system development life-cycle. However, the combination of Scrum, XP and agile patterns does not mean that they can directly be used to develop embedded systems. Slightly changes were needed to: *(i)* adopt processes and tools to optimize the product's design rather than take paths that lead to designs that have no chance of satisfying the constraints, *(ii)* support software and hardware development through a comprehensive flow from specification to implementation, *(iii)* instantiate the system platform based on the application constraints rather than overdesign a platform instance for a given product, and *(iv)* use system platform to conduct various design space exploration analyses for performance.

To illustrate the use of the processes and tools of the proposed methodology, we described how it could be applied to develop the pulse oximeter equipment. In this case study, we used UML notation to specify the application functionalities and convert them into programming languages to apply the estimation and partitioning tools. For further steps, we are researching models that can carry enough information about the ultimate physical implementation at a high abstraction level and planning experimental studies where the methodology will be observed. After that, our goal is to introduce the proposed methodology step-by-step into the industry by using traditional measurement framework.

7 Acknowledgements

This work is partially supported by Brazilian Council of Research CNPq under grant number 55.3164/2005-8.

References

[1] K. Beck and C. Andres. *Extreme Programming Explained - Embrace Change*. Second Edition, Addison-Wesley, 1999.

[2] S. Berczuk and B. Appleton. *Software Configuration Management Patterns*. First Edition, Addison-Wesley, 2002.

[3] J. O. Coplien and D. Schmidt. *Organizational Patterns of Agile Software Development*. First Edition, Prentice Hall, 2004.

[4] D. Gajski, F. Vahid, and S. Narayan. A system-design methodology: Executable-specification refinement. *European Conference on Design Automation, Paris, France*, 1994.

[5] B. Greene. Agile methods applied to embedded software development. *Proceeding of the Agile Development Conference (ADC'04).*, 2004.

[6] P. Koopman. Embedded system design issues (the rest of the story). *Proceedings of the International Conference on Computer Design (ICCD96)*, pages 310–317, 1996.

[7] P. Kukkala, J. Riihimki, M. Hnnikinen, T. Hmlinen, and K. Kronlf. Uml 2.0 profile for embedded system design. *Proceedings of the Design, Automation and Test in Europe Conference and Exhibition (DATE'05).*, 2005.

[8] P. Manhart and K. Schneider. Breaking the ice for agile development of embedded software: An industry experience report. *Proceedings of the 26th International Conference on Software Engineering (ICSE04)*, page 3647, 2004.

[9] K. Nguyen, Z. Sun, , and P. Thiagarajan. Model-driven soc design via executable uml to systemc. *Proceedings of the 25th IEEE International Symposium on Real-Time Systems, Page(s) 459-468*, pages 459–468, 2004.

[10] M. Oliveira. *Desenvolvimento de Um Protótipo para a Medida Não Invasiva da Saturação Arterial de Oxigênio em Humanos - Oxímetro de Pulso*. Master thesis, Center for Informatics at Federal University of Pernambuco, 2002.

[11] M. J. Oliveira, S. Neto, P. Maciel, R. Lima, A. Ribeiro, R. Barreto, E. Tavares, and F. Braga. Analyzing software performance and energy consumption of embedded systems by probabilistic modeling: An approach based on coloured petri nets. *ICATPN 2006, LNCS 4024, pp. 261281, 2006.*, page 261281, 2006.

[12] K. Schwaber and M. Beedle. *Agile Software Development with Scrum*. First Edition, Series in Agile Software Development, Prentice Hall, 2002.

[13] A. S. Vicentelli and G. Martin. Platform-based design and software design methodology for embedded systems. *IEEE Design and Test of Computers*, 18(6):23–33, 2001.

Embedded System Modeling based on Resource-Oriented Model

Jin Hyun Kim and Jin-Young Choi
Dept. of Computer Science, Korea University,
Seoul, 136-701, Republic of Korea
{jhkim,choi}@formal.korea.ac.kr

Abstract

In Model-Based Development(MBD), developers analyze, validate, implement, and test a system with based on the model of system. In the development of embedded system, MBD becomes more popular since the complexity of embedded system has been increasing nowadays. However, the produced software through MBD often diverges from the original model and even results in an erroneous situation because the model of system may not properly reflect the principal aspect of system. In this paper, we provide a hardware model to capture the property and constraint of hardware in embedded systems. In addition, we propose a development framework for embedded systems, in which the model of system is oriented to the model of hardware called resource model.

1 Introduction

An embedded system is a special-purpose system in which the software system is completely encapsulated by hardware device it controls. The hardware and the software components in it process various types of external inputs in an interactive manner. In recent, an embedded system is often deployed into the safety-critical systems, such as nuclear, avionics, automobile, that require higher quality and reliability than general computer systems. Moreover, the complexity of such systems requires rigorous and flexible methodologies like Model-Based Development(MBD). In MBD, the use of various dimensional models across all development phases allows the model-based design to produce systems that are more correct by model construction. The model in MBD often allows to verify itself by formal verification methods so that it gets more guaranteed than by simulation and testing.

However, embedded software produced from the model of embedded system more often diverges from the original model because the principal property or constraint of hardware, operating systems, and system software may be ignored by designers or analyzers.

In this paper, we present a hardware-oriented model as a key model that captures the property and constraint of hardware that restricts the behavior of software. In general, resource is an entity that software needs in order to perform their functions. The resource can often indicate various types of hardware, such as memory, CPU, and ports. Moreover, the resource often means an software entities, such as semaphore, message-queue, mailbox, and data packet,that are used to control software program and to process input data. To design embedded systems, we regard resource as such hardware and software entities. In addition, we incorporate functions to control the hardware and software entities into a part of resource model. With based on the resource model, we present a development framework for embedded systems. In our approach, we assume that hardware is first constructed, software runs on such hardware.

This paper is organized as follows. First, we will discuss the related works and problems in the embedded system model of MBD, particularly, focusing on the implementation. Next, we define embedded system resource in terms of embedded software and present a development method that is based on the resource model. After that, a case study is given to explain the resource-oriented method in developing and implementing embedded software. Finally, we conclude this paper and present the future work.

2 Related Works

To develop embedded systems, one of development approaches is codesign[2, 6, 7, 8]. There are two kinds of codesign;for different domains and for hardware and software. The first codesign approach[2] suggests that different domain models, such as continuous, discrete-event, and communicating finite state machine, synchronous dataflows, an so on, are co-designed into a model of heterogenous system by CFSM(Co-design Finite State Machine). In this approach, the validation of system model is often performed by simulation to confirm these models in different domains are correctly and consistently operate each of func-

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

tions. The second codesign approach[7, 8] emphasizes the digital systems for electronic systems only in view of hardware and software and suggests that an embedded system is designed without discriminating hardware and software in a system model, and then, the model of system is partitioned into hardware and software, and they are implemented respectively into hardware and software system. Stephan[8] also presented a hardware/software co-design framework where developers create models of a formal system representation independently of the hardware and software implementation. In his approach, the hardware and software are developed from each model of hardware and software. The main assumption of this paper is that hardware environment in the embedded system to be developed is given. So that our idea proposing in the paper is how to abstract the hardware in terms of resource model in order that application software can be developed easily in cooperation with the abstraction model. In this sense, our idea can be used in the codesign process once after hardware and software are partitioned. When application software is identified and implemented, one is needed for the interface between application software and hardware. The resource model can provide the view point of hardware for the software.

Another embedded system development approach is Platform-Based Design(PBD)[9]. In PBD, the platform is *a library of components* that can be assembled to generate a design at that level of abstraction. The library not only contains computational block but also communication components that are used to interconnect the functional components. In PBD, the essential principle is to identify precisely defined layers where the refinement and abstraction process take place and to identify design as a meeting-in-the-middle process, where successive refinement of specification meet with abstraction of potential implementation. In our approach, we focus on the abstraction of hardware, where hardware is captured in terms of specific properties and constraints, in order to guide the construction of software.

Development methodology using UML(Unified Modeling Language) is another approach for the development of embedded system. UML is a set of modeling languages that allow the user to prospect systems in various aspects[1]. However, in UML, it is necessary to construct more than one model to reason an embedded system, and it is difficult to construct models in each of views and to transform a model into another model. In our approach, we use a behavioral langauge, statechart[3, 4], to construct hardware and software model.

3 Comments on Embedded System Model

In MBD, there are various models, such as behavioral, architectural, structural model[1] to build embedded systems. Behavioral models, such as statechart and activity-diagram model, are widely used to capture an embedded system because they are able to be validated by executable models and generated into software or hardware implementation codes without human interpretation. In this paper, we use the statechart model to capture the hardware and software of embedded system.

A typical difference between hardware and software in embedded systems is the concept of concurrency. Hardware component, such as analogue, digital and dedicated hardware component, can perform each of functions in purely concurrent manner. However, software component in forms of tasks, processes, interrupt handlers, functions, and objects, can performs each of functions in interleaving manner. Thus, the middleware like operating systems is used to compensate such different concept of concurrency, but the accuracy of software in performing its function are closely related to the timing accuracy of software execution. In particular, the timing information about the behavior of hardware must be given in software design models in order to build more correct embedded software since such timing constraint placed on the software comes from the hardware.

In MBD, an embedded software construction requires a reworking of the implementation process[5]. This reworking of the process leads to

- A reduction in the amount of hand written code
- Improved modeling of the physical constraints of the control hardware
- Implementation issues being considered in the design process

The first reworking is related to the human interpretation of models. To reduce the human interpretation, the automatic code generation technique has been studied for last decades, and the size of automatically generated code is also properly reduced nowadays so that the produced code is used in the safety-critical system such as nuclear, avionic system.

In this paper, we focus on the last two since those are related to hardware constraints. The hardware constraints must be explicitly given in software requirement or design. To incorporate the hardware constraints into the software design, we present a hardware-oriented model called resource model, in which the property and constraint of hardware are essential for embedded software construction. The implementation issues must be also considered when the system model is constructed. In our hardware-oriented model. we fucus on the timing and the availability issue of hardware among various implementation issues.

To explain the necessity of resource model, we give a simple example where two tasks communicate with each other as follows.

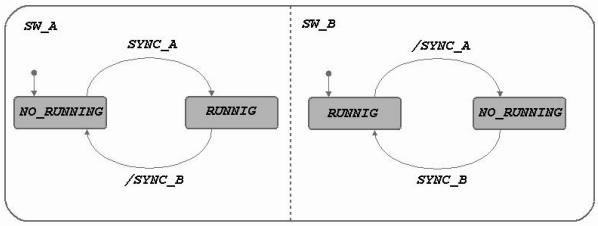


Figure 1. A Statechart Model for Two Tasks Synchronization

> **"Two tasks communicate through two events, SYNC_A and SYNC_B"**

Figure 1 shows a statechart model for two-task synchronization. The model in statechart depicts that two parts in the system are synchronizing with each other using events. To implement them into software, developers must consider the aspect of resource that is restricted in software environment. Figure 2 shows the software code of two-task synchronization, where each part of statechart models, SW_A and SW_B, are implemented into software in forms of task. Then, one of principal issues is synchronization method that controls software execution flow. For instance, developers might determine the following issues related to the software execution.

- Among Task A and Task B, which first starts its execution?
- What kind of semaphore is used to control the software execution if the synchronization method is implemented with semaphore?

These issues are mainly related to the characteristic of synchronizing method and the scheduling policy of CPU.

```
task A{
  while(1){
      sem_pend(sync_a);// NO_RUNNING
      ...// RUNNING
      sem_post(sync_b);
}

task B{
  while(1){
  sem_post(sync_a);
  ... // RUNNING
  sem_pend(sync_b);// NO_RUNNING
}
```

Figure 2. Two Tasks in Software Code

If developers determine to use semaphore for the synchronization, the semaphore may be characterized as follows.

- The type of semaphore is binary semaphore.
- Semaphore, SYNC_A, has be already acquired by another Task B, before Task A starts.
- The semaphore can not be preempted by any other tasks while it has been already preempted once.
- Function, sem_pend(), forces tasks to wait for the semaphore in waiting-list when the semaphore has been already acquired by a task.
- Function, sem_post(), force a task to ready for execution if the task are waiting for the semaphore in the waiting-list and to request CPU scheduling.

We would incorporate these constraints and requirements that are mostly related to the hardware of embedded system into a resource model. Thus, design specification in respect to embedded software can be detailed more correctly in terms of hardware that interacts with target software. Next, we characterize a resource to represent hardware in embedded systems.

4 Resource Model

In this paper, we characterize the resource of embedded system as follows:

> **Resource in embedded systems is a hardware-related object that is used for software execution.**

We divide the resource of embedded system into hardware-oriented and software-oriented resource. The software and hardware components, except target hardware and target software, may compose an embedded system. The hardware component is often divided into general-purpose hardware such as CPU, memories, AD/DA converters, digital I/Os, serial ports, and bus interfaces and dedicated hardware, such as FPGA, sensors, and motors. All of them often interact with software to accomplish a system requirement continually. These hardware components are hardware-oriented resources in embedded systems. The embedded software is often divided into application and system software. The system software is used to interpret hardware information for software. The application software uses such elements as semaphore, message-queue, and mailbox, that are served by an operating system, in order to control software execution and uses data that are transmitted by an operating system in order to process data. Thus, we incorporate such software elements the system software

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

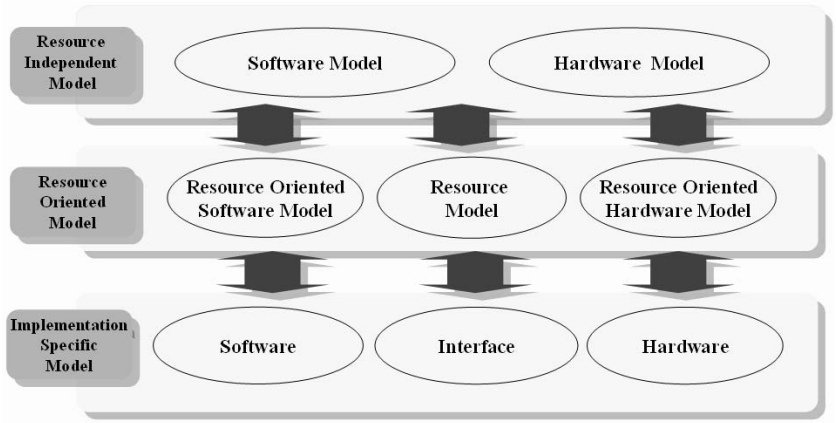


Figure 3. Development based on Resource-Oriented Model

like operating systems provides into software-oriented resource.

In our approach, the system software, such as operating system and device drivers, is regarded as methods to control hardware and software resources.

In our view, resource model is characterized as follows:

> **Resource model captures hardware in terms of constraints concerning hardware behavior, timing, and availability.**

The behavior of hardware is hardware action or function to restrict and support software behavior. The behavior of hardware to interact with embedded software is implemented in forms of *data* and *time*. The data highlight hardware status, hardware input value, and the composition of data and time controls the software execution flow. To produce correct embedded software models, we capture the correct behavior of hardware and implement them in consistent ways.

Next, the timing constraint related to hardware restricts the behavior of software, and the timing constraint is specified in forms of the deadline and the throughput. An embedded software must complete in a specific time when a hardware component expects the software to produce outputs for the hardware component. Therefore, most of embedded software is subject to the timing constraint of hardware, and such constraints should be introduced in embedded software model for software verification.

Finally, the availability in using a hardware unit such as memory and ports, is identified before software with deadline processes its function. Thus, it is necessary for the availability to be introduced in embedded software models. For instance, the memory in computer systems is a typical shared entity that more than one function share during their execution. The point is that the shared memory must never be corrupted by an illegal access. Thus, the access rule must be described to protect such shared resource from being corrupted by an illegal access, Hence, we believe that the availability of resource must be introduced in embedded software models.

The resource in our approach plays a role as requirement and constraint of hardware that is placed on the software. However, we discuss that the resource model is not only used for the verification of a model but also for the implementation of embedded software.

In our approach, the resource model to abstract hardware in embedded system has following advantages.

- The model of hardware that is co-designed with software model is limited to the constraint of hardware for embedded software construction.
- The co-designed model reduces the complexity of verifying a large specification.
- The model of hardware helps developers construct and configure interfaces, such as operating systems, interface data and functions.

5 Resource-Oriented Embedded System Development

Figure 3 shows a development of embedded system that is based on resource model. We divide the model of embedded systems model into three kinds of models; Resource-Independent Model(RIM), Resource-Oriented Model(ROM), and Implementation Specific Model(ISM).

Resource-Independent Model(RIM) captures embedded system models with focusing on a system behavior, not considering the use of any resources. This model is oriented to the state and communication of each function of systems. In Figure 1, a statechart model for the two-task

synchronization is depicted in the view of RIM. The RIM captures a system in terms of the orthogonal relation of functions so that it allows users to capture the behavior of each function in terms of the communication of the function in the overall system view. In statechart models, a state in statechart implies two kinds of means; action and property. *Action state* indicates system behavior in a particular time so it means something behavioral that happens in a particular time. *Property state*, by contrast, indicates system property, condition, or mode. Surely, the property may imply the notion of system action, but we separate the action notion from the property notion. For example, a statement, "The system is processing input A," implies an action of the system. In contrast, a statement, "The system is off," implies a condition of the system that is interpreted by the property state. The action state can be refined into a detailed statecharts while the property state cannot be refined into a lower hierarchy statechart level. To construct the RIM, we can first identify the action and the property state in a system specification. After that, the identified states are used to construct the behavior of a system while identifying the communication of each software components.

Resource-Oriented Model(ROM) constructs the model of embedded software where a resource model is incorporated into the RIM so that the constraint and requirement of hardware is placed on the software model. In other words, the ROM captures the behavior of embedded software that is oriented to the behavior of hardware because the resource model includes the property of hardware.

Figure 4 depicts an example of ROM where two tasks synchronize with each other using two semaphores. In particular, it runs two tasks in one CPU so there a scheduling algorithm is necessary to share it. The task of the ROM must process its function in accordance with constraints that is placed by the behavior and limitation of resource, and the behavior and limitation of resource is modeled in the resource model.

Consider a situation in which the embedded software can be restricted by the following selective requirements concerning semaphores.

- If the semaphore is binary, it is impossible for any other tasks to preempt the semaphore.
- If the semaphore is a counting semaphore, it can be shared by a limited number of tasks.

Figure 4 shows that the behavior of task depends on the property of semaphore. If the semaphore to be used by the two tasks is a counting semaphore, tasks may not be synchronized with each other. Moreover, this uses only one CPU, it needs a scheduling policy that determines the execution sequence of tasks.

Figure 4 specifies the requirement for the semaphore as follows:

- The type of semaphore is a binary semaphore, thus, it is exclusively used by only one task.
- The semaphore informs semaphore-requesting tasks of being released whenever the semaphore is released,
- The semaphore is marked with being preempted and allows semaphore-acquiring tasks to be re-scheduled whenever a task acquires the semaphore.

The mean that a software model is oriented to resource models is that the software model executes collaboratively with resource models that specify the property and constraint of hardware, and the property and constraint of hardware are closely related to the behavior of software. In statechart model of Figure 4, resource models, such as SEM_A, SEM_B, and SCHEDULER, are communicating with tasks, SW_A and SW_B using events and condition data, such as A_READY, B_READY, and SCHED.

In the system level design, the ROM can be a detailed design for embedded software because it includes the detailed information for software implementation. Moreover, in implementation stage, a resource model can be used to implement and configure interfaces, such as interrupt handlers and operating systems.

Implementation Specific Model(ISM) is an implementation model, in which the details of a system, such as data structure, and variable type, are given in a behavioral and static form. The ISM is divided into Dynamic ISM and Static ISM.

Dynamic ISM is a detailed behavioral description for the code of software functions. The model of statechart in such tools as STATEMATE MAGNUM and Rhapsody of I-Logix can be automatically and directly generated into software applications in C, C++, and JAVA without human interpretation. Especially, these tools for software implementation provides the aspect of software programming languages, such as type system, structure, even platform configurations, that allows users to develop software without user's code implementation. Thus, we regard models in statechart as a Dynamic ISM because it includes the detailed description of a function in the implementation level. In a case of software, the Dynamic ISM is configured in forms of task, process, interrupt handler, and object. The hardware of embedded systems is also implemented into various types of components by transforming the model of hardware into Hardware Description Language(HDL), such as VHDL, Verilog, Esterel, etc. In a case of the resource model, more is complex in implementing it than the model of the software and the hardware. The resource model can be transformed into the hardware-software interfaces, such

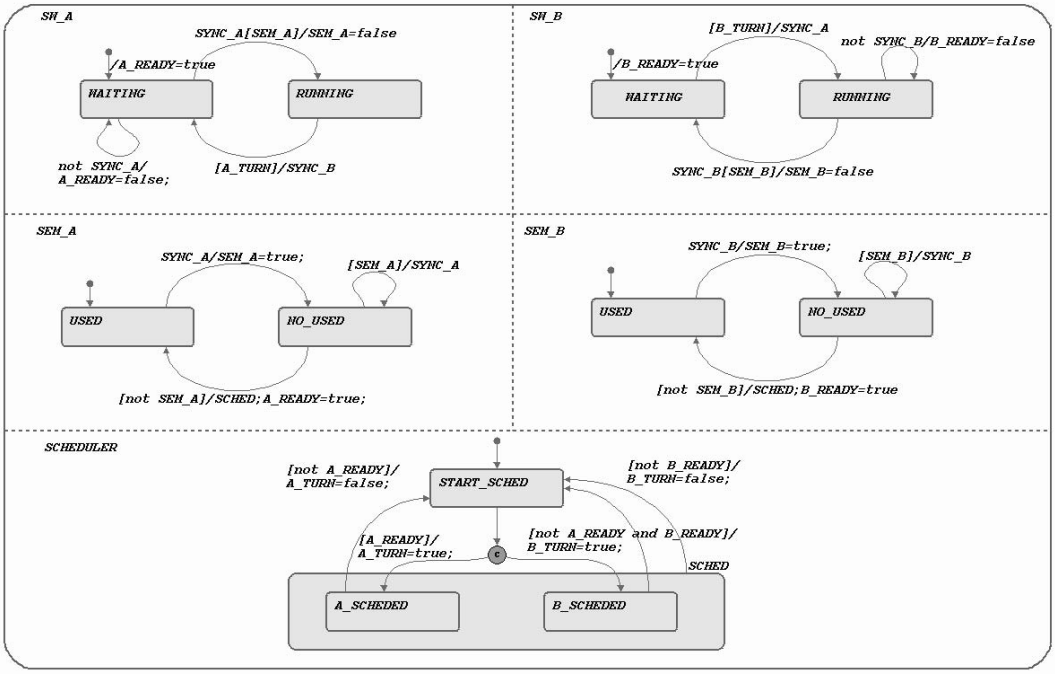


Figure 4. An Example of Resource-Oriented Model

as device drives, and operating systems, and it can be used to configure the property of operating systems, such as the scheduling policy, periodic scheduling time, and inter-task communication methods. In Figure 4, each of statecharts for task can be generated into software in C, C++, or JAVA. Thus, they are Dynamic ISMs.

Static ISM is a model that includes information about the interface data between the hardware and software of embedded systems. In embedded systems, the data to interpret the hardware information is implemented by software variables. To identify the needed data variables for the hardware and software interface, a resource model can be used because it represent the behavior of hardware with restricting the behavior of software and interacts with the model software. In the interacting of resource model with software models, the variables represent the interface data is identified. Thus, the interface data structure is captured into Static ISM. For instance, a resource model representing semaphore includes events, conditions, states, and data in statechart. The syntactical elements of statechart used to compose the resource model can be structured as follows:

```
STRUCT RSC{
    EVENT        e;
    CONDITION    c;
    DATA         d;
    STATE        s;
}
```

The abstract data types, EVENT, CONDITION, DATA, and STATE in the Dynamic ISM are implemented respectively into the corresponding data type by a dedicated software programming language. For instance, a variable in the type of CONDITION can be implemented by the type of boolean in JAVA. A variable in the type of STATE to identify the status of a system can realized by the type of integer in C, and a variable in the type of DATA can be realized into specific types of variable according to the use of the variable in models. The variables in the type of EVENT is needed to be specially treated in implementation environments. An event is often an instantaneous signal that is valid during only a specific period. For instance, the event in statechart only is valid in only one step. However, the imperative language, such as C, JAVA, C++, does not have the explicit notion of event, and it implements the event by event handling functions, such as event handler, and interrupt handler, that operating systems provide to handle such events. In short, the implementation of such events differs in user's implementation.

Consider a situation where where a variable in type of EVENT is implemented by an interrupt. To handle the interrupt, it is necessary to create an interrupt handler and some variables in implementation.

The point is that a resource model highlights the hardware of embedded system in terms of behavior, time, and availability. The resource model provides the property or constraint of hardware that restricts the behavior of software

with interacting with the software model in the resource-oriented model of embedded system.

Through the resource model, the model of embedded software is more detailed in terms of hardware constrains for the software. Moreover, the resource model is used to construct hardware and software interface and to configure middle-ware like operating systems and device drivers because it interprets the behavior of hardware in term of software.

In the development of embedded systems, the advantages in using the resource-oriented models are as follows:

1. In requirement analysis, the analyzer can overview a system in the overall behavior of system, not in the view of system data.
2. In design of system, a resource model captures the property and constraint of hardware of embedded systems, and the embedded software model is oriented on the resource model. The detailed design of software can be analyzed more correctly and completely against hardware properties.
3. In implementation of system, a resource model is used to build various types of interfaces, such as data structures, device drivers, and operating systems .
4. The verification using resource models is more reliable rather than using independent software models and more consistent through various models of embedded software. Moreover, the model of software including the property and constraint of hardware by resource models can be regarded as a co-designed model, and the co-designed model for software is more possible to be applied with formal verification techniques rather than the classical hardware/software codesign models.

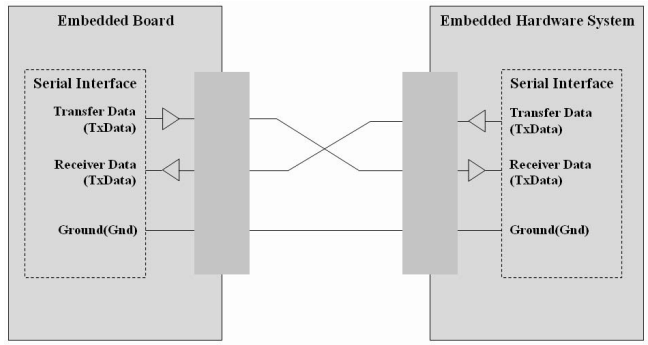
6 CASE STUDY: Serial Communication using UART

6.1 RIM for UART

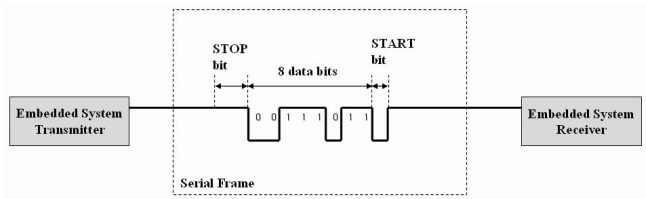
UART(Universal Asynchronous Receiver/Transmitter) is a full duplex serial interface that implements an asynchronous serial communications between a target system and a host system. The asynchronous communication informs a receive of data of the begging and the ending of the data transmission by sending either starting or ending bit. The serial communication using UART is illustrated in Figure 6.

```
STRUCT  DS_SEM_A{
        EVENT    SYNC_A;
        EVENT    SCHED;
        DATA     SEM_A;
        DATA     A_READY;
        STATE    ST_SEM_A;
}
```

Figure 5. Static ISM for Semaphore



(a) UART External Structure



(b) An Example of Serial Communication

Figure 6. Serial Communication with UART

The UART controller and its software is required as follows.

- The size of buffer of Transmitter/Receiver is 8-bit
- The START and STOP bit marks a frame with respectively the begging and the ending of a data frame.
- The STOP bit brings a serial interrupt to the system.
- A receiver reads the data from a buffer whenever it receives the serial interrupt from the UART controller.

The RIM for the UART controller and its software is depicted in Figure 7. The model specifies a concurrent execution of hardware and software; BUFFER_CONTROLLER and RX_SW_FUNCTION. The BUFFER_CONTROLLER specifies a UART controller that reads START_BIT to begin to receive a frame of data from a sender and STOP_BIT to stop to receive the

frame of data. The BUFFER_CONTROLLER also informs the receiver of the buffer status by sending an interrupt, RX_BUFFER_FULL_INTERRUPT. The software function, RX_SW_FUNCTION, processes the data transited from the buffer when invoked by an hardware interrupt, RX_BUFFER_FULL_INTERRUPT.

6.2 ROM for UART

The ROM of UART provides two timing constraints as follows:

- To discriminate the value of input signal, a timing constraint is placed on the UART.
- To limit the time when the software reads data from the buffer, a timing constraint related to buffer update is placed on the software. That is, the software must read the data from buffer before a new data of next frame arrives at the buffer.

The ROM in Figure 8 shows that the behavior of UART where it changes its state according to the value of data with timing constraints.

The BUFFER_CONTROLLER in Figure 8 is a statechart model for the UART controller. The RX_BUFFER is a resource to represent a shared buffer, and the size of buffer, 8 bits, is specified by a data variable, RX_BUF_LEFT. The size of buffer is reduced while the buffer is filled up with data. The controller looks forward to the arrival of the STOP_BIT when the buffer of controller is full of receiving data. In this model, The START_BIT and the STOP_BIT means into respectively 0 and 1, The only one of O value signal indicates the begging of receiving of a frame and two consecutive 1's indicates the ending of receiving of a frame.

The value of buffer, RX_BUFFER, in the state of BUFFERING is changed by signal the RX_SIG that denotes a value signal from a data sender. The rest

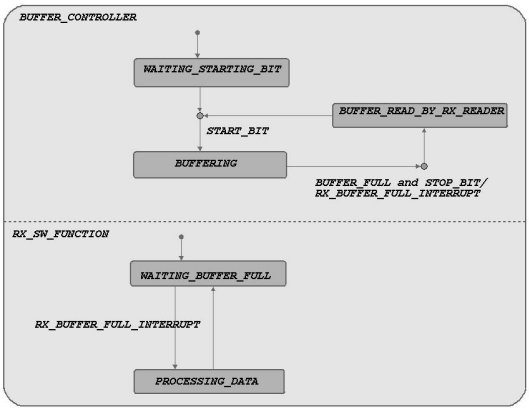


Figure 7. RIM for UART

of data in the RX_BUFFER is shifted 1 bit to left whenever a new data delivered by RX_SIG is put into the RX_BUFFER(ASHL(RX_BUFFER,1)). The full of RX_BUFFER leads RX_BUFFER to be ready for receiving two consecutive 1's, and after they all arrive at the buffer, the controller of buffer brings an serial interrupt, RX_BUFFER_FULL_INTERRUPT, in order to invoke a receiving software. The interrupt starts RX_SW_FUNCTION that processes the data of receiving by copying them into the processing software.

The ROM of the UART includes some timing requirements. The timing specification in the ROM is based on discrete time model. The UART in Figure 8 interacts with a time indicating signal, TICKER, that triggers a clock tick to inform UART of time. Hence, the UART can changes its behavior with periodic clock, TICK, that is occurred by the TICKER.

The following is one of the timing requirements that is related to UART:

> **To discriminate the value of input signal, a timing constraint is placed on the UART..**

The signal, RX_SIG, delivers the value of input signal. Time to capture the value of RX_SIG is controlled by a tick, TICK, that is triggered by the TICKER. In other words, the value of RX_SIG is sampled each occurrence of the tick. The model of UART shows that the RX_SIG always comes along with TICK as follows:

> **TICK**[not **RX_SIG** and **RX_BUF_LEFT**=0]/
> **RX_BUFFER=ASHL(RX_BUFFER,1)**

The following is another timing requirement that is for data processing software:

> **To limit the time when the software reads data from the buffer, a timing constraint related to buffer update is placed on the software. That is, the software must read the data from buffer before a new data of next frame arrives at the buffer.**

This requirement shows that the data processing software must copy the data of RX_BUFFER into a software variable, DATA, within a specific period. The software function starts when the software is synchronized by the serial an interrupt that indicates the ending of transmission of a data frame. The timing requirement shows that the software to copy the data of RX_BUFFER into its software variable before a new START_BIT is arrived at the buffer of UART. The sender of UART must know the period between the STOP_BIT and the START_BIT in order to allow software to perform the process of the copy. The following statement of statechart in Figure 8 shows condition and time when the UART starts

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

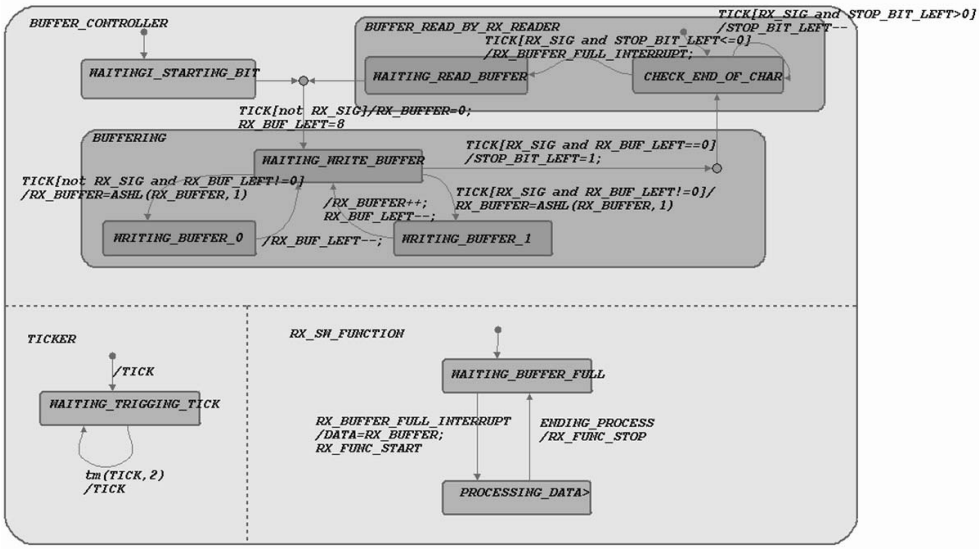


Figure 8. ROM for UART

again to receive data from the sender. The data processing software must complete to copy the data of buffer before the the next TICK arrives.

TICK[not **RX_SIG**]/
RX_BUFFER=0;RX_BUF_LEFT=8;

6.3 ISM for UART

The implementation model for UART is based on the ROM of UART, which is mentioned in the previous section. First, we construct a Static ISM for the UART. The Static ISM for the interface of hardware and software is constructed through, a resource model for UART, BUFFER_CONTROLLER. The data structure for interfacing data may be organized as follows:

```
STRUCT UART{
    RX_BUFFER     : Bit-array (0 to 7);
    RX_BUF_LEFT   : Integer;
    STOP_BIT_LEFT : Integer;
}
```

We assume that two events, TICK and RX_BUFFER_FULL_INTERRUPT is implemented in two hardware interrupts. Moreover, we create interrupt handlers to handle the two interrupts and to synchronize tasks that are related to the interrupts.

7 Resource-Oriented Development

In our approach, resource models capture the property and constraints of hardware in embedded systems. The development framework of embedded systems, which is based on the resource model, is shown in Figure 9.

To analyze a system requirement, developers focus on the behavior and the communication of system in terms of the overall system using the RIM. Next, a resource model is constructed in that it captures the property and constraints of hardware that is oriented to the behavior, the time, and the availability of hardware. Using the resource model, embedded software model is detailed with restricted by the property and constraint of hardware. A resource model often reveals the property of interfaces, such as operating systems, device drivers. In verification of model of embedded systems, resource models are used as requirement specification for embedded software. In other words. the model of embedded software is verified by checking if the behavior of embedded software model satisfies the constraints of hardware and are compliance with the property of hardware.

The ISM is used to implement software product in that the application of software and the interface are constructed using the model of software and the resource model. However, the resource can be used to configure operating systems, interrupt handlers, and device drivers if these system software are already determined in the development process.

8 Conclusions

This paper presents a resource model for the modeling of embedded system. MBD allows developers to specify and validate a system without physical implementation of system. However, embedded software may diverge from the original software model due to such physical constraint

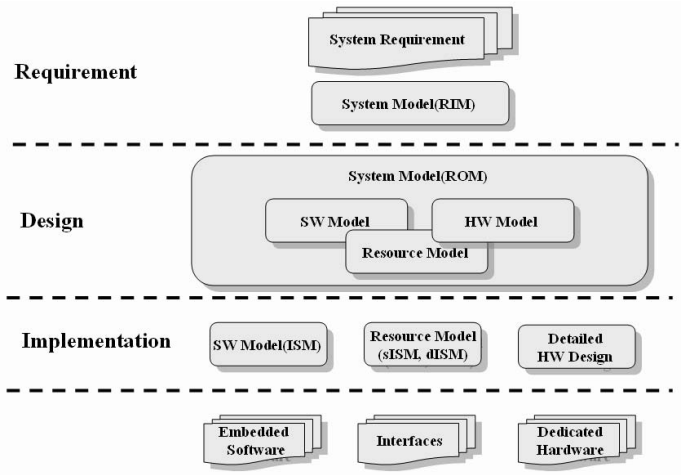


Figure 9. Resource-Oriented Embedded System Development

as hardware, operating systems, system software and interfaces.

We present a resource model that highlights the property and constraints of hardware in terms of embedded software. The resource model help developers in the development of embedded system as follows:

- A resource model of embedded system provides a requirement or constraint for embedded software.
- The model of embedded software is more correctly validated with based on more precisely hardware behavior model in a resource model.

We presented three kinds of design model for embedded system; Resource-Independent Model, Resource-Oriented Model, and Implementation Specific Model. They are used for respectively requirement, detained design, and implementation model. Moreover, we think that one of most important and difficult parts of application software development in embedded system is interface - the software's view to hardware environment. We think the notion of resource for hardware environment can be one of solutions for the hardware-software interface in software development

In the future, we will define resource model with formal semantics and develop ways to generate interfaces using resource model without human interpretation.

References

[1] Bruce Powel Douglass. *Real-time UML (2nd ed.): developing efficient objects for embedded systems*. Addison-Wesley Longman Ltd., Essex, UK, UK, 2000.

[2] S. Edwards, L. Lavagno, E. A. Lee, and A. Sangiovanni-Vincentelli. Design of embedded systems: Formal models, validation, and synthesis. *Proc. of the IEEE*, 85(3), year 1997.

[3] David Harel. Statecharts: A visual formalism for complex systems. *Sci. Comput. Program.*, 8(3):231–274, 1987.

[4] David Harel and Eran Gery. Executable object modeling with statecharts. *Computer*, 30(7):31–42, 1997.

[5] D Lamberson, N.P Teske, and J.K. Hedrick. A model-based approach to the implementation of automotive embedded control systems. In *Intelligent Transportation Systems, 2003. Proceedings. 2003 IEEE*, volume 1, pages 655–659, 2003.

[6] Alberto Sangiovanni-Vincentelli Marco Sgroi, Luciano Lavagno. Formal models for embedded system design. *IEEE Design and Test of Computers*, 17(2):14–27, June 2000.

[7] G. De Micheli and M. Sami. Hardware-software codesign, 1996.

[8] Stephan Schulz, Jerzy W. Rozenblit, Michael Mrva, and Klaus Buchenrieder. Model-based codesign. *Computer*, 31(8):60–67, 1998.

[9] Richard Zurawski. *Embedded Systems Handbook*. CRC Press, Inc., Boca Raton, FL, USA, 2004.

Diagnosis of Embedded Software using Program Spectra*

Peter Zoeteweij[1] Rui Abreu[1] Rob Golsteijn[2] Arjan J.C. van Gemund[1]

[1]Embedded Software Lab
Delft University of Technology
The Netherlands
{p.zoeteweij,r.f.abreu,a.j.c.vangemund}@tudelft.nl

[2]Innovation Center Eindhoven
NXP Semiconductors
The Netherlands
rob.golsteijn@nxp.com

Abstract

Automated diagnosis of errors detected during software testing can improve the efficiency of the debugging process, and can thus help to make software more reliable. In this paper we discuss the application of a specific automated debugging technique, namely software fault localization through the analysis of program spectra, in the area of embedded software in high-volume consumer electronics products. We discuss why the technique is particularly well suited for this application domain, and through experiments on an industrial test case we demonstrate that it can lead to highly accurate diagnoses of realistic errors.

Keywords: diagnosis, program spectra, automated debugging, embedded systems, consumer electronics.

1 Introduction

Software reliability can generally be improved through extensive testing and debugging, but this is often in conflict with market conditions: software cannot be tested exhaustively, and of the bugs that are found, only those with the highest impact on the user-perceived reliability can be solved before the release. In this typical scenario, testing reveals more bugs than can be solved, and debugging is a bottleneck for improving reliability. Automated debugging techniques can help to reduce this bottleneck.

The subject of this paper is a particular automated debugging technique, namely software fault localization through the analysis of *program spectra* [11]. These can be seen as projections of execution traces that indicate which parts of a program were active during various runs of that program. The diagnosis consist in analyzing the extent to which the activity of specific parts correlates with errors detected in the different runs.

Locating a fault is an important step in actually solving it, and program spectra have successfully been applied for this purpose in several tools focusing on various application domains, such as Pinpoint [4], which focuses on large, dynamic on-line transaction processing systems, AMPLE [5], which focuses on object-oriented software, and Tarantula [9], which focuses on C programs.

In this paper, we discuss the applicability of the technique to embedded software, and specifically to embedded software in high-volume consumer electronics products. Software has become an important factor in the development, marketing, and user-perception of these products, and the typical combination of limited computing resources, complex systems, and tight development deadlines make the technique a particularly attractive means for improving product reliability.

To support our argument, we report the outcome of two experiments, where we diagnosed two different errors occurring in the control software of a particular product line of television sets from a well-known international consumer electronics manufacturer. In both experiments, the technique is able to locate the (known) faults that cause these errors quite well, and in one case, this implies an accuracy of a single statement in approximately 450K lines of code.

The remainder of this paper is organized as follows. In Section 2 we explain the diagnosis technique in more detail, and in Section 3 we discuss its applicability to embedded software in consumer electronics products. In Section 4 we describe our experiments, and in Section 5 we discuss how our current implementation can be improved. In Section 6 we discuss related work. We conclude in Section 7.

2 Preliminaries

In this section we introduce program spectra, and describe how they are used for diagnosing software faults.

*This work has been carried out as part of the TRADER project under the responsibility of the Embedded Systems Institute. This project is partially supported by the Netherlands Ministry of Economic Affairs under the BSIK03021 program.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

```
void RationalSort(int n, int *num, int *den)
{  /* block 1 */
   int i,j,temp;

   for ( i=n-1; i>=0; i-- ) {
      /* block 2 */
      for ( j=0; j<i; j++ ) {
         /* block 3 */
         if (RationalGT(num[j], den[j],
                        num[j+1], den[j+1])) {
            /* block 4 */
            temp = num[j];
            num[j] = num[j+1];
            num[j+1] = temp; } } }
}
```

Figure 1. A faulty C function for sorting rational numbers

First we introduce the necessary terminology.

2.1 Failures, Errors, and Faults

As defined in [3], we use the following terminology.

- A *failure* is an event that occurs when delivered service deviates from correct service.
- An *error* is the part of the total state of the system that may cause a failure.
- A *fault* is the cause of an error in the system.

To illustrate these concepts, consider the C function in Figure 1. It is meant to sort, using the bubble sort algorithm, a sequence of `n` rational numbers whose numerators and denominators are passed via parameters `num` and `den`, respectively. There is a fault (bug) in the swapping code of block 4: only the numerators of the rational numbers are swapped. The denominators are left in their original order.

A failure occurs when applying `RationalSort` yields anything other than a sorted version of its input. An error occurs after the code inside the conditional statement is executed, while `den[j]` $\neq$ `den[j+1]`. Such errors can be temporary: if we apply `RationalSort` to the sequence $\langle \frac{4}{1}, \frac{2}{2}, \frac{0}{1} \rangle$, an error occurs after the first two numerators are swapped. However, this error is "canceled" by later swapping actions, and the sequence ends up being sorted correctly. Faults do not automatically lead to errors either: no error will occur if the input is already sorted, or if all denominators are equal.

The purpose of *diagnosis* is to locate the faults that are the root cause of detected errors. As such, error detection is a prerequisite for diagnosis. As a rudimentary form of error detection, failure detection can be used, but in software more powerful mechanisms are available, such as pointer checking, array bounds checking, deadlock detection, etc.

In a software context, faults are often called *bugs*, and diagnosis is part of *debugging*. Computer-aided techniques as the one we consider here are known as *automated debugging*.

2.2 Program Spectra

A program spectrum [11] is a collection of data that provides a specific view on the dynamic behavior of software. This data is collected at run-time, and typically consist of a number of counters or flags for the different parts of a program. As such, recording a program spectrum is a lightweight analysis compared to other run-time methods, such as, e.g., dynamic slicing [10].

As an example, a *block count spectrum* tells how often each block of code is executed during a run of a program. In this paper, a block of code is a C language statement, where we do not distinguish between the individual statements of a compound statement, but where we do distinguish between the cases of a switch statement[1]. Suppose that the function `RationalSort` of Figure 1 is used to sort the sequence $\langle \frac{2}{1}, \frac{3}{1}, \frac{4}{1}, \frac{1}{1} \rangle$, which it happens to do correctly. This would result in the following block count spectrum, where block 5 refers to the body of the `RationalGT` function, which has not been shown in Figure 1.

block	1	2	3	4	5
count	1	4	6	3	6

Block 1, the body of the function `RationalSort`, is executed once. Blocks 2 and 3, the bodies of the two loops, are executed four and six times, respectively. To sort our example array, three exchanges must be made, and block 4, the body of the conditional statement, is executed three times. Block 5, the `RationalGT` function body, is executed six times: once for every iteration of the inner loop.

If we are only interested in whether a block is executed or not, we can use binary flags instead of counters. In this case, the block count spectra revert to block *hit* spectra. Beside block count/hit spectra, many other forms of program spectra exist. See [7] for an overview. In this paper we will work with block hit spectra, and hit spectra for logical threads used in the software of our test case (see Section 4.1).

2.3 Fault Diagnosis

The hit spectra of M runs constitute a binary matrix, whose columns correspond to N different parts of the program (see Figure 2). In our case, these parts are blocks of

[1] This is a slightly different notion than a *basic block*, which is a block of code that has no branch.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

$$
\begin{array}{c}
 & N \text{ parts} & & & & \text{errors} \\
M \text{ spectra} & \begin{bmatrix} x_{11} & x_{12} & \dots & x_{1N} \\ x_{21} & x_{22} & \dots & x_{2N} \\ \vdots & \vdots & \ddots & \vdots \\ x_{M1} & x_{M2} & \dots & x_{MN} \end{bmatrix} & & & \begin{bmatrix} e_1 \\ e_2 \\ \vdots \\ e_M \end{bmatrix} \\
 & s_1 \quad s_2 \quad \dots \quad s_N & & &
\end{array}
$$

Figure 2. The ingredients of fault diagnosis

C code. In some of the runs an error is detected. This information constitutes another column vector, the error vector. This vector corresponds to a hypothetical part of the program that is responsible for all observed errors. Fault localization essentially consists in identifying the part whose column vector resembles the error vector most.

In the field of data clustering, resemblances between vectors of binary, nominally scaled data, such as the columns in our matrix of program spectra, are quantified by means of *similarity coefficients* (see, e.g., [8]). As an example, the Jaccard similarity coefficient (see also [8]) expresses the similarity s_j of column j and the error vector as the number of positions in which these vectors share an entry 1 (i.e., block was exercised and the run has failed), divided by this same number plus the number of positions in which the vectors have different entries:

$$s_j = \frac{a_{11}(j)}{a_{11}(j) + a_{01}(j) + a_{10}(j)} \tag{1}$$

where $a_{pq}(j) = |\{i \mid x_{ij} = p \wedge e_i = q\}|$, and $p, q \in \{0, 1\}$.

Under the assumption that a high similarity to the error vector indicates a high probability that the corresponding parts of the software cause the detected errors, the calculated similarity coefficients rank the parts of the program with respect to their likelihood of containing the faults.

To illustrate the approach, suppose that we apply the `RationalSort` function to the input sequences $I_1 = \langle\ \rangle$, $I_2 = \langle \frac{1}{4} \rangle$, $I_3 = \langle \frac{2}{1}, \frac{1}{1} \rangle$ and $I_4 = \langle \frac{4}{1}, \frac{2}{2}, \frac{0}{1} \rangle$, $I_5 = \langle \frac{3}{1}, \frac{2}{2}, \frac{4}{3}, \frac{1}{4} \rangle$, and $I_6 = \langle \frac{1}{4}, \frac{1}{3}, \frac{1}{2}, \frac{1}{1} \rangle$.

I_1, I_2, and I_6 are already sorted, and lead to passed runs. I_3 is not sorted, but the denominators in this sequence happen to be equal, in which case no error occurs. I_4 is the example from Section 2.1: it is not sorted, and an error occurs during its execution, but this error goes undetected. Only for I_5 the program fails. The calculated result is $\langle \frac{1}{1}, \frac{2}{2}, \frac{4}{3}, \frac{3}{4} \rangle$ instead of $\langle \frac{1}{4}, \frac{2}{2}, \frac{4}{3}, \frac{3}{1} \rangle$, which is a clear indication that an error has occurred.

The block hit spectra for these runs are as follows ('1' denotes a hit), where block 5 corresponds to the body of the `RationalGT` function, which has not been shown in Figure 1.

	block					
input	1	2	3	4	5	error
I_1	1	0	0	0	0	0
I_2	1	1	0	0	0	0
I_3	1	1	1	1	1	0
I_4	1	1	1	1	1	0
I_5	1	1	1	1	1	1
I_6	1	1	1	0	1	0

For this data, the calculated Jaccard coefficients are $s_1 = \frac{1}{6}$, $s_2 = \frac{1}{5}$, $s_3 = \frac{1}{4}$, $s_4 = \frac{1}{3}$, $s_5 = \frac{1}{4}$, which (correctly) identifies block 4 as the most likely location of the fault.

3 Relevance to Embedded Software

The effectiveness of the diagnosis technique described in the previous section has already been demonstrated in several articles (see, e.g., [1], [4], [9]). In this paper we present the benefits and discuss the issues specifically related to debugging embedded software in consumer electronics products. Especially because of constraints imposed by the market, the conditions under which this software is developed are somewhat different from those for other software products:

- To reduce unit costs, and often to ensure portability of the devices, the software runs on non-commodity hardware, and computing resources are limited.

- As a consequence, many facilities that developers of non-embedded software have come to rely on are absent, or are available only in rudimentary forms. Examples are profiling tools that give insight in the dynamic behavior of systems.

- At the same time, the systems are highly concurrent, and operate at a low level of abstraction from the hardware. Therefore, their design and implementation are complicated by factors that can largely be abstracted away from in other software systems, such as deadlock prevention, and timing constraints involved in, e.g., writing to the graphics display only in those fractions of a second that the screen is not being refreshed.

- On top of challenges that the entire software industry has to deal with, such as geographically distributed development organizations, the strong competition between manufacturers of consumer electronics makes it absolutely vital that release deadlines are met.

- Although important safety mechanisms, such as short-circuit detection, are sometimes implemented in software, for a large part of the functionality there are no personal risks involved in transient failures.

Consequently, it is not uncommon that consumer electronics products are shipped with several known software faults outstanding. To a certain extent, this also holds for other software products, but the combination of the complexity of the systems, the tight constraints imposed by the market, and the relatively low impact of the majority of possible system failures creates a unique situation. Instead of aiming for correctness, the goal is to create a product that is of value to customers, despite its imperfections, and to bring the reliability to a commercially acceptable level (also compared to the competition) before a product must be released.

The technique of Section 2 can help to reach this goal faster, and may thus reduce the time-to-market, and lead to more reliable products. Specific benefits are the following.

- As a black-box diagnosis technique, it can be applied without any additional modeling effort. This effort would be hard to justify under the market conditions described above. Moreover, concurrent systems are difficult to model.
- The technique improves insight in the run-time behavior. For embedded software in consumer electronics, this is often lacking, because of the concurrency, but also because of the decentralized development.
- We expect that the technique can easily be integrated with existing testing procedures, such as overnight playback of recorded usage scenarios. In addition to the information that errors have occurred in some scenarios, this gives a first indication of the parts of the software that are likely to be involved in these errors. In the large, geographically distributed development organizations that we are dealing with, it may also help to identify which teams of developers to contact.
- Last but not least, the technique is light-weight, which is relevant because of the non-commodity hardware and limited computing resources. All that is needed is some memory for storing program spectra, or for calculating the similarity coefficients on the fly (which reduces the space complexity from $O(M \times N)$ to $O(N)$, see Section 5). Profiling tools such as `gcov` are convenient for obtaining program spectra, but they are typically not available in a development environment for embedded software. However, the same data can be obtained through source code instrumentation.

While none of these benefits are unique, their combination makes program spectrum analysis an attractive technique for diagnosing embedded software in consumer electronics.

4 Experiments

In this section we describe our experience with applying the techniques of Section 2 to an industrial test case.

4.1 Platform

The subject of our experiments is the control software in a particular product line of analog television sets. All audio and video processing is implemented in hardware, but the software is responsible for tasks such as decoding remote control input, displaying the on-screen menu, and coordinating the hardware (e.g., optimizing parameters for audio and video processing based on an analysis of the signals). Most teletext[2] functionality is also implemented in software.

The software itself consists of approximately 450K lines of C code, which is configured from a much larger (several MLOC) code base of Koala software components [12].

The control processor is a MIPS running a small multitasking operating system. Essentially, the run-time environment consists of several threads with increasing priorities, and for synchronization purposes, the work on these threads is organized in 315 logical threads inside the various components. Threads are preempted when work arrives for a higher-priority thread.

The total available RAM memory in consumer sets is two megabyte, but in the special developer version that we used for our experiments, another two megabyte was available. In addition, the developer sets have a serial connection, and a debugger interface for manual debugging on a PC.

4.2 Faults

We diagnosed two faults, one existing, and one that was seeded to reproduce an error from a different product line.

Load Problem. A known problem with the specific version of the control software that we had access to, is that after teletext viewing, the CPU load when watching television (TV mode) is approximately 10% higher than before teletext viewing. This is illustrated in Figure 3, which shows the CPU load for the following scenario: one minute TV mode, 30 s teletext viewing, and one minute of TV mode. The CPU load clearly increases around the 60th sample, when the teletext viewing starts, but never returns to its initial level after sample 90, when we switch back to TV mode.

Teletext Lock-up Problem. Another product line of television sets provides a function for searching in teletext pages. An existing fault in this functionality entails that searching in a page without visible content locks up the teletext system. A likely cause for the lock-up is an inconsistency in the values of two state variables in different components,

[2] A standard for broadcasting information (e.g., news, weather, TV guide) in text pages, very popular in Europe.

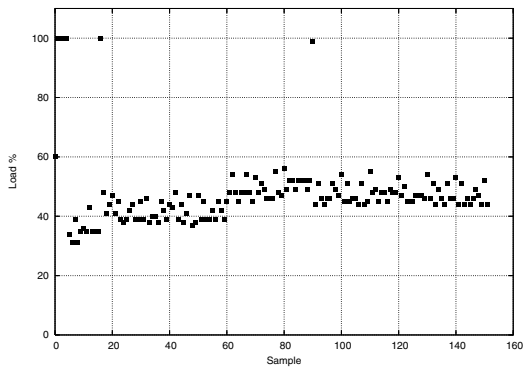


Figure 3. CPU load measured per second

for which only specific combinations are allowed. We hard-coded a remote control key-sequence that injects this error on our test platform.

4.3 Implementation

We wrote a small Koala component for recording and storing program spectra, and for transmitting them off the television set via the serial connection. The transmission is done on a low-priority thread while the CPU is otherwise idle, in order to minimize the impact on the timing behavior. Pending their transmission via the serial connection, our component caches program spectra in the extra memory available in our developer version of the hardware.

For diagnosing the load problem we obtained hit spectra for the logical threads mentioned in Section 4.1, resulting in spectra of 315 binary flags. We approached the lock-up problem at a much finer granularity, and obtained block hit spectra for practically all blocks of code in the control software, resulting in spectra of over 60,000 flags.

The hit spectra for the logical threads are obtained by manually instrumenting a centralized scheduling mechanism. For the block hit spectra we automatically instrumented the entire source code using the Front [2] parser generator.

In Section 2.3 we use program spectra for different runs of the software, but for embedded software in consumer electronics, and indeed for most interactive systems, the concept of a run is not very useful. Therefore we record the spectra per *transaction*, instead of per run, and we use two different notions of a transaction for the two different faults that we diagnosed:

- for the load problem, we use a periodic notion of a transaction, and record the spectra per second.
- for the lock-up problem, we define a transaction as the computation in between two key-presses on the remote control.

4.4 Diagnosis

For the load problem we used the scenario of Figure 3. We marked the last 60 spectra, for the second period of TV mode as 'failed,' and those of earlier transactions as 'passed.' In the ranking that follows from the analysis of Section 2.3, the logical thread that had been identified by the developers as the actual cause of the load problem was in the second position out of 315. In the first position was a logical thread related to teletext, whose activation is part of the problem, so in this case we can conclude that although the diagnosis is not perfect, the implied suggestion for investigating the problem is quite useful.

For the lock-up problem, we used a proper error detection mechanism. On each key-press, when caching the current spectrum, a separate routine verifies the values of the two state variables, and marks the current spectrum as failed if they assume an invalid combination. Although this is a special-purpose mechanism, including and regularly checking high-level assert-like statements about correct behavior is a valid means to increase the error-awareness of systems.

Using a very simple scenario of 23 key-presses that essentially (1) verifies that the TV and teletext subsystems function correctly, (2) triggers the error injection, and (3) checks that the teletext subsystem is no longer responding, we immediately got a good diagnosis of the detected error: the first two positions in the total ranking of over 60,000 blocks pointed directly to our error injection code. Adding another three key-presses to exonerate an uncovered branch in this code made the diagnosis perfect: the exact statement that introduced the state inconsistency was located out of approximately 450K lines of source code.

5 Discussion

Especially the results for the lock-up problem have convinced us that program spectra, and their application to fault diagnosis are a viable technique and useful tool in the area of embedded software in consumer electronics. However, there are a number of issues with our implementation.

First, we cannot claim that we have not altered the timing behavior of the system. Because of its rigorous design, the TV is still functioning properly, but everything runs much slower with the block-level instrumentation (e.g., changing channels now takes seconds). One reason is that currently, we collect block *count* spectra at byte resolution, and convert to block *hit* spectra off-line. Updating the counters in a multi-threaded environment requires a critical section for every executed block, which is hugely expensive. Fortunately, this information is not used, and we believe we can implement a binary flag update without a critical section.

Second, we cache the spectra of passed transactions, and transmit them off the system during CPU idle time. Be-

cause of the low throughput of the serial connection, this may become a bottleneck for large spectra and larger scenarios. In our case we could store 25 spectra of 65,536 counters, which was already slowing down the scenarios with more than that number of transactions, but even with a more memory-efficient implementation, this inevitably becomes a problem with, for example, overnight testing.

For many purposes, however, we will not have to store the actual spectra. In particular for fault diagnosis, ultimately we are only interested in the calculated similarity coefficients, and all similarity coefficients that we are aware of are expressed in terms of the four counters a_{00}, a_{01}, a_{10}, and a_{11} introduced in Section 2.3. If an error detection mechanism is available, like in our experiments with the lock-up problem, then these four counters can be calculated on the fly, and the memory requirements become linear in the number columns in the matrix of Figure 2.

6 Related Work

Program spectra were introduced in [11], where hit spectra of intra-procedural paths are analyzed to diagnose year 2000 problems. The distinction between count spectra and hit spectra is introduced in [7], where several kinds of program spectra are evaluated in the context of regression testing. In the introduction we already mentioned three practical diagnosis/debugging tools [4, 5, 9] that are essentially based on the same diagnosis method as ours. A recent study, reported in [1], indicates that the choice of the similarity coefficient, as introduced in Section 2.3 can be of significant influence on the quality of the diagnosis. In the experiments reported in the present paper we used both the Jaccard coefficient of Eq. (1), and the best coefficient identified in [1], but the results were essentially the same.

As we mentioned in Section 3, black box techniques like spectrum-based diagnosis can be applied without additional knowledge about a system. An example of a white box technique is model-based diagnosis (see, e.g., [6]), where a diagnosis is obtained by logical inference from a formal model of the system, combined with a set of run-time observations. White box approaches to software diagnosis exist (see, e.g., [13]), but software modeling is extremely complex, so most software diagnosis techniques are black box.

7 Conclusion

In this paper we have demonstrated software fault diagnosis through the analysis of program spectra, on a large-scale industrial test case in the area of embedded software in consumer electronics devices. In addition to confirming established effectiveness results, our experiments indicate that the technique lends itself well for application in the resource-constrained environments that are typical for the development of embedded software.

While our current experiments focus on development-time debugging, they open corridors to further applications, such as run-time recovery by rebooting only those parts of a system whose activities correlate with detected errors.

8 Acknowledgments

We would like to thank Pierre van de Laar for valuable comments on an earlier version of this paper.

References

[1] R. Abreu, P. Zoeteweij, and A. J. C. van Gemund. An evaluation of similarity coefficients for software fault localization. In *Proceedings of PRDC'06*, pages 39–46. IEEE Computer Society, 2006.

[2] L. Augusteijn. Front: a front-end generator for Lex, Yacc and C, release 1.0, 2002. See `http://front.sourceforge.net/`.

[3] A. Avižienis, J.-C. Laprie, B. Randell, and C. E. Landwehr. Basic concepts and taxonomy of dependable and secure computing. *IEEE Trans. Dependable Sec. Comput.*, 1(1):11–33, 2004.

[4] M. Y. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer. Pinpoint: Problem determination in large, dynamic internet services. In *Proceedings of DSN 2002*, pages 595–604. IEEE Computer Society, 2002.

[5] V. Dallmeier, C. Lindig, and A. Zeller. Lightweight defect localization for Java. In A. P. Black, editor, *Proceedings of ECOOP 2005*, volume 3586 of *LNCS*, pages 528–550. Springer-Verlag, 2005.

[6] J. de Kleer and B. C. Williams. Diagnosing multiple faults. *Artif. Intell.*, 32(1):97–130, 1987.

[7] M. J. Harrold, G. Rothermel, R. Wu, and L. Yi. An empirical investigation of program spectra. *ACM SIGPLAN Notices*, 33(7):83–90, 1998.

[8] A. K. Jain and R. C. Dubes. *Algorithms for clustering data*. Prentice-Hall, Inc., 1988.

[9] J. A. Jones, M. J. Harrold, and J. Stasko. Visualization of test information to assist fault localization. In *Proceedings of ICSE 2002*, pages 467–477. ACM Press, 2002.

[10] B. Korel and J. Laski. Dynamic program slicing. *Information Processing Letters*, 29:155–163, 1988.

[11] T. Reps, T. Ball, M. Das, and J. Larus. The use of program profiling for software maintenance with applications to the year 2000 problem. In M. Jazayeri and H. Schauer, editors, *Proceedings of ESEC/FSE 97*, volume 1301 of *LNCS*, pages 432–449. Springer–Verlag, 1997.

[12] R. van Ommering, F. van der Linden, J. Kramer, and J. Magee. The Koala component model for consumer electronics software. *IEEE Computer*, March 2000.

[13] F. Wotawa, M. Strumptner, and W. Mayer. Model-based debugging or how to diagnose programs automatically. In T. Hendtlass and M. Ali, editors, *Proceedings of IAE/AIE 2002*, volume 2358 of *LNCS*, pages 746–757. Springer-Verlag, 2002.

Integrating Security Modeling into Embedded System Design

Matthew Eby, Jan Werner, Gabor Karsai, Akos Ledeczi
Institute for Software Integrated Systems
Vanderbilt University, Nashville, TN 37235
{firstname.lastname}@vanderbilt.edu

Abstract

There is an ever increasing concern about security threats as embedded systems are moving towards networked applications. Model based approaches have proven to be effective techniques for embedded systems design. However, existing modeling tools were not designed to meet the current and future security challenges of networked embedded systems. In this paper, we propose a framework to incorporate security modeling into embedded system design. We've developed a security analysis tool that can easily integrate with existing tool chains to create co-design environments that addresses security, functionality and system architecture aspects of embedded systems concurrently.

1. Introduction

Embedded systems play a crucial role in critical infrastructure in [1], which is essential to national security, success and economic health [2]. There is increasing concern of the security threats on these kinds of embedded systems [3],[4]. Successful attacks have been reported on the US Power Grid [5] and the sewer system of Australia's Maroochy Shire Council [6]. Other incidents such as a worm infection [7] have affected the Davis-Besse Nuclear Power Plant and CSX Railroad Corp. [6]. To address such security threats we need to rethink the embedded software design process.

Model Integrated Computing (MIC) [8] is gaining wide recognition in the field of embedded software design. Models represent embedded software, its deployment platform and its interactions with the physical environment. Models facilitate formal analysis, verification, validation and generation of embedded systems [9]. Hence, this approach is superior to traditional manual software development process. Although, there is modeling tool support for analysis of functionality, performance, power consumption, safety, etc., currently available tools incorporate little if any support for security modeling. As a result, security is looked at only once the complete system has been built. At best, this approach of addressing security in the last stages of development is inefficient taking large amounts of effort to achieve only modest improvements in security. Engineers designing embedded systems usually do not have the experience to address security issues and in many cases are not even aware of the issues [10]. Fixing security vulnerabilities involves releasing patches, which can introduce new problems such as viruses [11] or security vulnerabilities [12]. Still, systems designed without security in mind are intrinsically insecure. Patches can fix specific security vulnerabilities, but do not address poor system architecture.

Many times vulnerabilities are only discovered once they have been exploited. To address these unknown threats, systems can be isolated in private corporate networks using firewalls and intrusion detection systems. But such perimeter defenses, even if they are flawless, cannot protect against insider attacks [13]. In light of this situation, we advocate modeling environments that incorporate security into the early design phase of embedded systems.

One of the few modeling languages with security extensions is UML. Currently available extensions to UML provide: access control [14] [15], fair exchange, assumptions about secrecy, integrity, etc. [16]. There are existing tools that can guarantee some security properties using automated proof verifiers [16]. However, UML based Model Driven Security is not sufficient for design and analysis of embedded systems. We strongly believe that embedded system design can benefit from Domain-Specific Modeling Languages (DSML) as opposed to the one-size-fits-all approach of UML. For example, there is no concept of hardware in UML which makes it ill suited for embedded systems with their diverse hardware architectures. In many embedded applications system resources are scarce. Added overhead for security can

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

have drastic effects on performance. An ideal embedded software development environment will allow the engineer to analyze security and performance tradeoffs based on the hardware platform the system will run on.

2. Background and Motivation

MIC can meet the challenges of designing secure embedded systems. A key advantage of the model based approach is the abstraction of the application domain. This abstraction is facilitated through the use of DSMLs. A DSML provides a system designer a set of concepts that are specifically tailored for a certain application domain. In our case, the domain is networked embedded real-time systems, such as process control systems, automotive, avionics and robotics systems. A DSML with the proper level of abstraction hides the inconsequential details of a system while allowing the engineer to shift focus to more important aspects. There are many examples of DSMLs developed for embedded system design in different domains [MILAN [17], SMOLES [18], AADL [17]]. We propose an extension mechanism for DSMLs that adds security concepts similar to UML extensions [14]. By extending embedded system DSMLs, we can add tool support for security analysis, validation, verification and generation. These security tools will extend the large tool chains that already exist for embedded system design.

3. General Approach

We will demonstrate a process for integrating security analysis into existing tool chains to create a security co-design environment. The approach taken is to create a common DSML that is used to capture and analyze security properties of systems. The advantage of this approach is that the effort needed develop the security analysis tool is only spent once. Then this tool can be incorporated into existing embedded systems languages with minimal effort. By defining mappings from an embedded system DSML onto the security analysis DSML, we can analyze the security properties the embedded system. Figure 1 illustrates the process of defining mappings from one or more DSMLs onto a language supporting security analysis and feeding the analysis results back to the DSML.

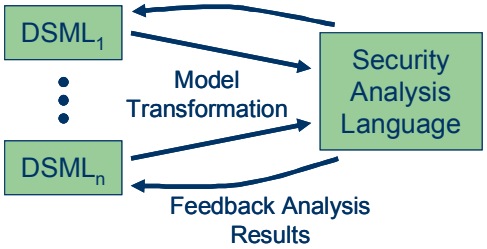


Figure 1. Mappings from DSMLs to SAL enable security analysis of the DSMLs

The co-design environment is implemented in the Generic Modeling Environment (GME) [9]. GME is a metaprogrammable tool which facilitates the graphical implementation of DSMLs through the use of metamodels. In this environment, we create a Security Analysis Language (SAL) that enables a user to model and analyze security related properties of embedded systems. (Note that while SAL is technically a DSML, from this point out we use the term DSML only in reference to a language for embedded systems design which we wish to add security analysis capabilities to.) The purpose of this analysis tool is to identify points in the system model that violate certain security requirements and provide useful feedback to the modeler. SAL allows such violations to be identified and remedied at design time before they can be exploited. Currently, SAL supports two types of analyses: information flow analysis and threat model analysis, which are detailed in the following sections.

3.1. Information Flow Analysis

The two traditional models for dealing with information flow in systems are the Bell-LaPadula model [19] and the Biba model [20]. Both of these models enforce an access control scheme that defines the rights of a subject to access information. Subjects and information are assigned a security level and a compartment which define what information a given subject is permitted to access. The set of all security levels is an ordered set that can be evaluated as an inequality (i.e. Top Secret > Secret). Compartments are a set that can be evaluated as an inequation (i.e. FBI ≠ CIA).

The Bell-LaPadula model deals with confidentiality or secrecy of information in systems. The two properties that the Bell-LaPadula enforces are the Simple Security Property and the * (star) Property. The Simple Security Property states that a subject may not read information that is classified at a level greater than that subject's classification. The * Property states that a subject may not write information to a level less than that subject's classification.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

The Biba model deals with integrity of information in systems. Biba also enforces a Simple Security Property and a * Property. The Biba version of the Simple Security Property states that a subject may not read information that is classified at a level less than that subject's classification and the Biba version of the * Property states that a subject may not write information to a level greater than that subject's classification.

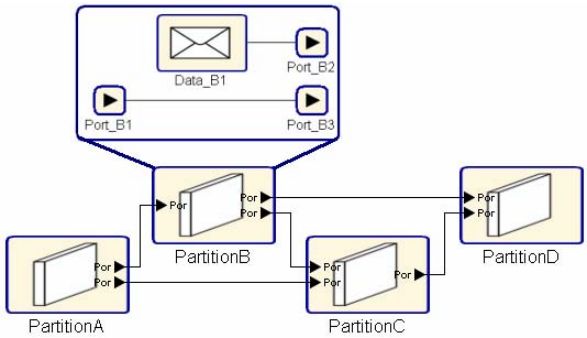


Figure 2. Partitions and dataflows in SAL

To analyze the Bell-LaPadula and Biba models, SAL views a system as a set of partitions, a set of data objects contained in each partition and the dataflows inside and across the partitions. Dataflows are represented as connections between input and output ports on a partition. In SAL, partitions are the subjects and are assigned a *security level* and *compartment* attributes. A data object inherits the *security level* and *compartment* classification of its containing partition. SAL allows the *security level* to be an integer value and the *compartment* to be a string value. Our analysis tool treats each data object as the root node in a tree search algorithm. The tool will traverse the dataflow paths originating from a data object and verify that each partition through which that data object flows has a *security level* and *compartment* that permit that partition to access the data object. Bell-LaPadula does not allow information to flow to a lower *security level* while Biba does not allow information to flow to a higher *security level*. When composed, these two models only allow information to flow between partitions with the same *security level*. Applying both models is too restrictive in a system where the designer does not need to restrict access to all data objects. There may be some data objects that have a secrecy requirement but no integrity requirement and vice versa. To provide a less restrictive model, data objects in SAL are assigned two Boolean attributes, *secrecy* and *integrity*. The flow of every data object is evaluated based on the settings of these attributes. When *secrecy* is true the Bell-LaPadula model is enforced and when *integrity* is true the Biba model is enforced on the flow of that data object between partitions. Figure 2 shows a small example model in SAL.

3.2. Threat Model Analysis

In a distributed system, partitions may reside on multiple nodes and data is transferred between these nodes over some communication channel. The information flow analysis addresses the movement of data explicitly defined in the system model but does not address covert channels. One such covert channel could be a man-in-the-middle attack on the communication channel. To prevent such attack, the communication channel can be encrypted. Adversary modeling in SAL enables the analysis tool to identify vulnerable channels and determine which encryption algorithms can be used to protect data being transmitted on that channel. Figure 3 illustrates the adversary model.

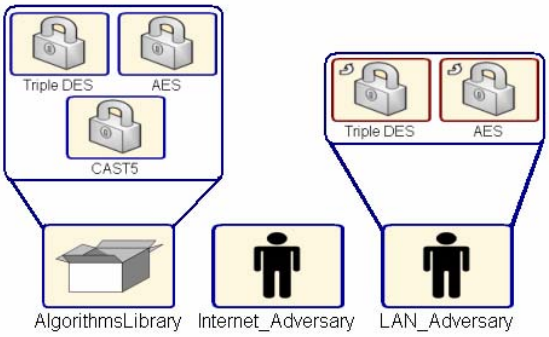


Figure 3. Encryption algorithms library and adversary models in SAL

In each system there is a library of encryption algorithms that contains the set of all encryption algorithms that can be used to encrypt a channel. Each system also contains a set of adversary models that define which encryption algorithms are vulnerable in the context of that adversary. Each adversary contains a set of references to algorithms that are defined in the algorithms library. Each reference has an attribute, *maxkeysize*, which means that the referenced algorithm is vulnerable to that adversary if the strength of its encryption is not greater than *maxkeysize*. Together, the encryption algorithm library and adversary models allow our analysis tool to determine which algorithms are safe to use to encrypt information flows. Each information flow in SAL has an attribute, *adversary*, which identifies the adversary model associated with that information flow. Each information flow in SAL also has an *EncryptionAlgorithm* and *KeySize* attribute. For each information flow in the system, the analysis

tool checks the *EncryptionAlgorithm* and *KeySize* attribute against the set of encryption algorithms that are vulnerable for the adversary model specified by *adversary*.

3.3. Integrating Security Analysis with Existing Tool Chains

Although, there is modeling tool support for analysis of functionality, performance, power consumption, safety, etc., currently available tools incorporate little if any support for security modeling. As a result, security is only addressed once the complete system has been built. We want to leverage the work behind existing tool chains by incorporating security analysis in the system design process. SAL was created to be a reusable tool that can be integrated with multiple tool chains, thus reducing the effort that would be required to develop custom security analysis for each tool chain.

In MIC, a transformation is, in general, a one way function with a domain being the set of all valid models in the original DSML and a range being the set of all valid models in the destination DSML. By defining a transformation that maps models of an embedded system DSML onto SAL, we can perform information flow analysis and threat model analysis on the embedded systems models. In order to define such a transformation, the original DSML must be able to capture those security properties that are need for SAL to provide a useful analysis. In other words, for information flow analysis, the DSML must be able to model the concepts such as data object, dataflow, partition, security level, compartment, secrecy and integrity requirements and for threat model analysis the DSML must be able to model the concepts such as encryption algorithm library, encryption algorithm, adversary model, vulnerable encryption algorithm, encrypted channel and channel adversary. Typically, the DSML will not have all of the concepts needed to create such a transformation. For example, take a DSML built on the synchronous dataflow model of computation [21]. This DSML would have the concepts such as data objects and dataflow, but none of the other security specific concepts. It would be the responsibility of a tool designer to add the ability to capture these security specific concepts in a DSML. The process of extending a DSML to capture security related properties is not as difficult as it might seem. One of the powerful concepts of the MIC approach is easy composition of metamodels to form new languages. By composing the metamodel of a DSML with concepts from SAL, it is relatively easy to form these security specific extensions to an existing language. The tool designer can then create the transformation rules that map models in the DSML onto models in SAL. GME is integrated with the Graph Rewriting and Transformation language (GReAT) [22]. GReAT is built on top of an execution engine (GReAT-E) which can translate models based on transformation rules specified by GReAT. This mapping specification needs to be created only once for a given DSML and then any valid models for that DSML can be automatically transformed into a corresponding SAL model.

Since SAL is only capable of capturing those concepts which are relevant to security analysis, it is not possible to define a transformation from SAL back onto the original DSML. Those concepts which are unique to the original domain are lost in the translation from the DSML to SAL. In order for SAL to provide useful feedback to the user, we introduce the *path* and *id* attributes which belong to partition, information flow, data object, ports, adversary model and encryption algorithm in SAL. *Path* and *id* store the path and the unique ID of an object in the original DSML. When a SAL model is analyzed and security violations are identified, the results will be fed back to the user of the original DSML in the form of an error messages along with hyperlinks that identify at which point in the original model there is a security violation. Using this approach, a user of a DSML will never have to view the SAL model. The transformation to a SAL model, the security analysis and the result feedback form an automated process.

When adding these security specific concepts to a DSML, it is important to consider what they mean in the context of the entire tool chain. Often the design flow includes other tools for such things as functionality, schedulability, power consumption and safety analysis. The division between these different types of analyses is not always clear-cut. Many times decisions made based on one type of analysis can have an impact on the outcome of other types of analysis. One such example in the context of SAL is the use of encryption algorithms. The decision to encrypt a communication channel could have a major effect on the schedulability of the system. Also, if there is a code generator for the DSML, it must be modified to support these security properties (i.e. linking to encryption libraries, enforcing the partition model, etc.). The tool developer who is integrating SAL capabilities to a DSML must address concerns such as making these other tools aware of the impact the security properties will have on the system. Figure 4 shows a typical design flow for performing security analysis with an embedded system DSML.

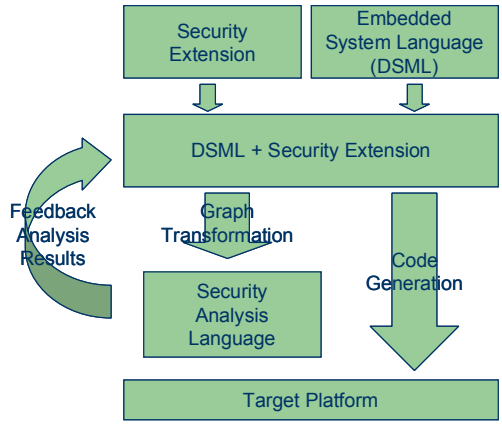


Figure 4. Typical embedded system design flow with SAL

4. Example: Integrating Security Analysis to an Existing Embedded System Language

As a proof of concept, we have integrated SAL with an existing tool for the design of embedded systems called SMOLES [18]. SMOLES alone does not address security concerns. We show how we add the capability for capturing security specific properties to SMOLES. Models in SMOLES are enriched with these concepts and we create a mapping from SMOLES concepts to objects in SAL. This mapping allows us to create a transformation from SMOLES models to SAL models. We show how results from the analysis of a SAL model can feed back warnings of security violations so that the user can correct the SMOLES model accordingly.

4.1. Description of SMOLES

The Simple Modeling Language for Embedded Systems (SMOLES) was designed as a simple modeling language that allows constructing small, embedded systems from components [18]. The components are assumed to be concurrently executing objects that communicate and synchronize with each other. Furthermore, objects can perform blocking I/O operations in which they wait for the result, while other objects can execute. Communication between components means passing data from a source component to a destination component, which is then enabled to run, in order to process the data. In addition to data triggering, periodic timers can also trigger the components. The language consists of components and assemblies. Components are the elementary building blocks, and contain input and output ports, which are used to receive/send data tokens from/to other components. Assemblies contain components, and describe how they are interconnected. Like components, assemblies can have their own input and output ports, and assemblies can contain other assemblies. Assemblies are organized into a hierarchy. The various components and assemblies in the hierarchy communicate with each other through the dataflows, as specified by the designer. SMOLES has a code generation utility that will interpret the model and output C++ code that will execute on top of a small custom dataflow kernel. Figure 5 shows a small example model in the SMOLES DSML.

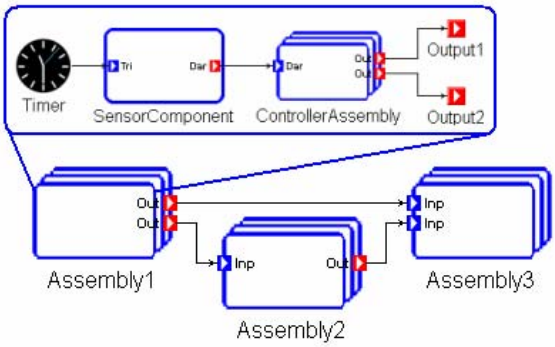


Figure 5. Example SMOLES model

4.2. Integrating SAL with SMOLES

Since SMOLES does not capture any security properties, we must add the appropriate concepts, so that we can define a transformation from SMOLES to SAL. We call this extended language SMOLES_SEC. SMOLES_SEC allows the modeler to capture the security properties required to perform the two types of analysis that SAL supports, the information flow analysis and threat model analysis.

First, we address those concepts necessary to perform the information flow analysis. SMOLES already has the concept of dataflows but none of the other concepts used in SAL. Assemblies in SMOLES are close to the concept of a partition in SAL. One possible approach would be to add the *security level* and *compartment* attributes to assemblies. However, assemblies are organized in a hierarchy whereas partitions in SAL are not. So, we introduce the idea of partition to SMOLES_SEC. Partitions will have input and output ports which can be connected by dataflow connections. Like in SAL, partitions will have *security level* and *compartment* attributes that define their access rights in the context of the Bell-LaPadula and Biba models. Assemblies will be contained by partitions and will inherit the *security level* and *compartment* of the containing partition. SMOLES has the concept of dataflow; however there is no first class object that is a data object. We assign the *secrecy* and

integrity attributes to an assembly in SMOLES_SEC and evaluate these attributes against the dataflows originating from that assembly.

Next, we address the concepts necessary to perform the threat model analysis. SMOLES has no concept of encryption algorithms or adversary modeling. We add these concepts to SMOLES_SEC and define them in the same way that they are defined for SAL. In each system, there is an encryption algorithms library with a set of encryptions algorithms. Each system contains a set of adversaries and each adversary contains a set of references to encryptions algorithms. Each encryption algorithm has an attribute, *maxkeysize*. We do not associate an encryption algorithm and adversary model directly with dataflows as it is done in SAL. Rather, SMOLES_SEC can model a deployment diagram where nodes, which represent the execution platform, are connected to other nodes through a link (or bus). A node can be viewed as a set that contain partitions that execute on that node. Likewise, a link can be viewed as a set that contains the dataflows that are transmitted over that link. Each link in SMOLES_SEC has an attribute, *adversary*, which identifies the adversary model associated with that link. Each link in SMOLES_SEC also has an *EncryptionAlgorithm* and *KeySize* attribute. Dataflows inherit the *adversary*, *KeySize* and *EncryptionAlgorithm* of the link that they are transmitted across.

The effect that these extensions for SMOLES_SEC have on tools that were written for the SMOLES languages must be examined. SMOLES has a utility that will generate C++ code from models. This utility is unaware of encryption algorithms and their meaning in the context of SMOLES_SEC. There are two solutions to this problem. Either we define a transformation that will map SMOLES_SEC models back onto SMOLES models or we port this code generator to work with the SMOLES_SEC environment. In our case we choose to port the code generator to the SMOLE_SEC environment. To do this we will need to make some slight modifications such as enabling assemblies to be contained in partitions and linking to a library of encryption algorithms.

4.3. Model Transformation from SMOLES_SEC to SAL

Now that the appropriate concepts have been added to extend SMOLE_SEC we are able to define a model transformation that maps SMOLES_SEC models to corresponding models in SAL. Once we have defined these rules, the process of converting SMOLES_SEC models to SAL models will be automated. We have written a small script that can be invoked from the SMOLES_SEC environment. This allows the user of the SMOLES_SEC environment to transform their model into a SAL model in one step, run the information flow and threat model analysis on the SAL model and receive the analysis results.

4.4. Example Application in SMOLES_SEC

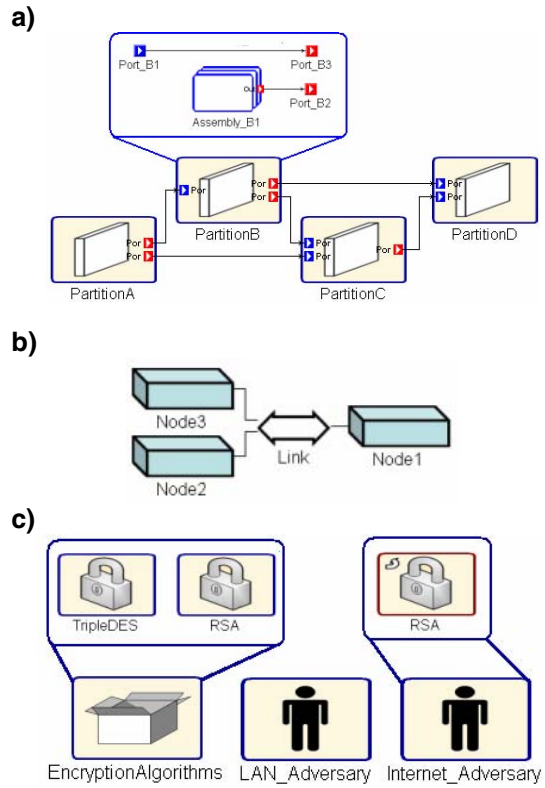


Figure. 6 SMOLES_SEC example application a) partitions and dataflows, b) deployment diagram, c) threat model.

An example SMOLES_SEC model is shown in Figure 6. This is a generic application that demonstrates the capabilities of the security analysis. Real applications of this tool will be too large to cover in the scope of this paper. There are four partitions in the system. Partitions A, B, and D have a *security level* of 1 and PartitionC has a *security level* of 2. This means that data objects with a *secrecy* requirement may not flow from PartitionC and those data object with an *integrity* requirement may not flow to PartitionC. In this example, we do not consider partitions with different *compartment* classifications. PartitionB contains an Assembly_B1 that has an *integrity* requirement but no *secrecy* requirement.

Since this assembly is in PartitionB it inherits the security level of 1. Figure 6b shows the deployment diagram. Nodes 1, 2, and 3 are connected by a common link. PartitionA and PartitionB execute on Node1. PartitionC executes on Node2 and PartitionD executes on Node3. All dataflows transfer data across the Link, except the dataflow connecting PartitionA and PartitionB which reside on the same node. Figure 6c shows the threat model of this system. The adversary model of Link is the Internet_Adversary.

First, we will invoke the information flow analysis on this model. Figure 7 shows the error message that we receive for the flow analysis. The assembly in PartitionB has an *integrity* requirement so the dataflows originating from this assembly are evaluated with the Biba model. There is a dataflow that connects PartitionB to PartitionC which represents data objects moving from a low security level to a high security level. This dataflow violates the Biba model. There are a several possible solutions to this error. It is up to the system modeler to determine which solution is appropriate in the context of their system. For this example, we determine that the PartitionB can be classified at a *security level* of 2. When this change is made to the model the security analysis tool does not return any errors.

```
Integrity Requirement Violated --
/SimpleSystem/PartitionB/Assembly_B1 has
an integrity requirement which is
violated by the information flow
connecting
/SimpleSystem/PartitionB/Port_B2 to
/SimpleSystem/PartitionC/Port_C1.
```

Figure 7 Error message - information flow analysis

Next, we invoke the threat model analysis. Figure 8a shows the error message we receive. There is an adversary associated with the link so the channel must be encrypted. The error message warns that Link must be encrypted so we set the encryption attributes on Link to 256 bit RSA. Internet_Adversary is capable of breaking RSA with a key size of 256 bits or less. The error message in Figure 8b shows the error message we receive. To fix this error message we increase the key size used to encrypt Link to 512 bits. This fixes the security violation the threat model analysis no longer returns any error messages.

```
a)  Connection not encrypted -- The
    connection at path: /SimpleSystem/Link is
    vulnerable. Please specify an encryption
    algorithm.

b)  Vulnerable Encryption Algorithm -- The
    connection at path: /SimpleSystem/Link is
    vulnerable. The adversary knows RSA up to
    a key size of 256 bits. Please specify a
    larger key size or change the algorithm.
```

Figure 8 Error messages – threat model analysis

5. Future Work

SAL currently supports modeling of access control policies in the context of the Bell-LaPadula and Biba models. These access control policies are not sufficient for the needs of all applications. We would like for SAL to have the expressiveness to model other types of access control schemes. Another area that needs to be addressed is how to more tightly integrate the security analysis with the other analysis tools available for a DSML. This would allow the designer to look at tradeoffs made based on security properties such as analyzing the tradeoffs between security and performance. We have shown how SAL can be integrated with SMOLES which is a dataflow based language. This leads to a simple mapping from SMOLES to SAL. There needs to be work done to look at how security analysis can be integrated with other classes of DSML such as those based on control flow.

6. Conclusion

Model driven security approaches have been successfully used in various industrial, governmental and financial applications. Model-Integrated Computing has proven to be a valuable tool in embedded systems design process. We have demonstrated a security analysis tool that is capable of analyzing the flow of data objects through a system and identifying points in a distributed system that are vulnerable to attack. We have outlined a method for composing this type of security tool with existing tool chains for DSMLs. This approach leverages the development efforts that have gone into design of tool suites for existing embedded system DSMLs. Creating a separate analysis language for security properties allows reuse of this tool for multiple DSMLs. The example application shown is a proof of concept that demonstrates the potential of integrating security modeling capabilities with existing languages.

7. Acknowledgement

This work was supported in part by TRUST (The Team for Research in Ubiquitous Secure Technology), which receives support from the National Science Foundation (NSF award number CCF-0424422).

8. References

[1] Fernandez, J. D. and Fernandez, A. E. 2005. *SCADA systems: vulnerabilities and remediation*. *J. Comput. Small Coll.* 20, 4 (Apr. 2005), 160-168.

[2] Amin, M. North America's electricity infrastructure: are we ready for more perfect storms? Security & Privacy Magazine, IEEE Volume 1, Issue 5, Sept.-Oct. 2003 Page(s):19 - 25

[3] Kocher, P., Lee, R., McGraw, G., and Raghunathan, A. 2004. Security as a new dimension in embedded system design. In *Proceedings of the 41st Annual Conference on Design Automation* (San Diego, CA, USA, June 07 - 11, 2004). DAC '04. ACM Press, New York, NY, 753-760.

[4] Ravi, S., Raghunathan, A., Kocher, P., and Hattangady, S. 2004. Security in embedded systems: Design challenges. *Trans. on Embedded Computing Sys.* 3, 3 (Aug. 2004), 461-491.

[5] F. Schneider, editor. *Trust in* .National Academy Press, Washington, DC, 1999. Available at http://www.nap.edu/readingroom/books/trust/

[6] Andrew Hildick-Smith, *Security for Critical Infrastructure SCADA systems.* August 24 2005. SANS Institute. Available at http://www.sans.org/rr/whitepapers/warfare/1644.php

[7] Weaver, N., Paxson, V., Staniford, S., and Cunningham, R. 2003. *A taxonomy of computer worms*. In *Proceedings of the 2003 ACM Workshop on Rapid Malcode* (Washington, DC, USA, October 27 - 27, 2003). WORM '03. ACM Press, New York, NY, 11-18

[8] Sztipanovits, J.; Karsai, G. *Model-integrated computing,* Computer Volume 30, Issue 4, April 1997 Page(s):110 – 111

[9] Karsai, G., Sztipanovits, J., Ledeczi, A., Bapty, T.: *"Model-Integrated Development of Embedded Software,"* Proceedings of the IEEE, Vol. 91, No.1., pp. 145-164, January, 2003

[10] Tsipenyuk, K.; Chess, B.; McGraw, G.; *Seven pernicious kingdoms: a taxonomy of software security errors* Security & Privacy Magazine, IEEE Volume 3, Issue 6, Nov.-Dec. 2005 Page(s):81 - 84

[11] Microsoft TechNet. Information About Virus-Infected Hotfixes. April 25 2001. Available at http://www.microsoft.com/technet/ security/alerts/info/vihotfix.mspx

[12] Microsoft TechNet. Microsoft Security Bulletin MS06-007. February 14 2006. Available at http://www.microsoft.com/technet /security/Bulletin/MS06-007.mspx

[13] Kevin Poulsen, *Slammer worm crashed Ohio nuke plant network,* August 19 2003. Available at http://www.securityfocus.com/ news/6767

[14] Jürjens, J. 2005. *Sound methods and effective tools for model-based security engineering with UML*. In *Proceedings of the 27th international Conference on Software Engineering* (St. Louis, MO, USA, May 15 - 21, 2005). ICSE '05. ACM Press, New York, NY, 322-331.

[15] Basin, D., Doser, J., and Lodderstedt, T. 2003. *Model driven security for process-oriented systems*. In *Proceedings of the Eighth ACM Symposium on Access Control Models and Technologies* (Como, Italy, June 02 - 03, 2003). SACMAT '03. ACM Press, New York, NY, 100-109.

[16] Jan Jürjens, *Secure Systems Development with UML*, Springer-Verlag, 2004

[17] Available from the Authors

[18] Szemethy, T. and Karsai, G. 2004. Platform modeling and model transformations for analysis. *Journal of Universal Computer Science 10*, 10, 1383–1406.

[19] D.E. Bell and L.J. LaPadula. "Secure Computer Systems: Mathematical Foundations and Model," Mitre Corp. Report No. M74-244, Bedford, Mass., 1975.

[20] K.J. Biba, "Integrity Considerations for Secure Computer Systems," Mitre Corp. Report TR-3153. Bedford. Mass., 1977.

[21] E. A. Lee and D. G. Messerschmitt, "Synchronous data flow," *Proc. IEEE*, vol. 75, pp. 1235–1245, Sept. 1987.

[22] Agrawal, A., Karsai, G., Ledeczi, A.: An End-to-End Domain-Driven Software Development Framework. Conference on Object Oriented Programming Systems Languages and Applications, 2003.

A Practical Approach for Process Family Engineering of Embedded Control Software

Cord Giese
Delta Software Technology
Eichenweg 16, D-57392 Schmallenberg, Germany
cgiese@d-s-t-g.com

Arnd Schnieders
Hasso-Plattner-Institute at the University of Potsdam
Prof.-Dr.-Helmert-Strasse 2-3, D-14482 Potsdam, Germany
schnieders@hpi.uni-potsdam.de

Jens Weiland
DaimlerChrysler Research and Technology
P.O.Box 2360, D-89013 Ulm, Germany
jens.weiland@daimlerchrysler.com

Abstract

This paper introduces an approach for the development of families of embedded control processes using process family engineering techniques, exemplarily in the automotive domain, based on UML. The feasibility of our approach is validated by means of a tool chain, which is applied to a windshield wiper case study.

1. Introduction

Today, up to 100 embedded electronic control units are typically found in premium cars on the market; and the number is still increasing [ScZ03]. Moreover, embedded control systems occur in many different variants, which allows for a better market penetration. Sources of variation in embedded control systems are manifold; for example the need to apply them in different markets (Europe, NAFTA, etc.) with their different regulations and customer-related requirements or the application of different electronical systems. Additionally, new software variants providing new features have to be brought to the market in short time intervals while at the same time a high software quality has to be assured.

In the long run these challenges can only be faced by advanced software development techniques like product family engineering applying code generators for the efficient assembly of product family members. However, up to now product family engineering research has concentrated on software systems, where structure diagrams like class diagrams or component diagrams represent the main blueprint for the development of the software system, while process-oriented software has not been regarded adequately.

As a result, existing product family engineering approaches are not suited well for the development of embedded control software, where processes describe possible consequences and the interplay of the execution of related control functions: sensor data received are processed through the control function and control signals are sent to the actuators as a result of the calculation (Figure 1).

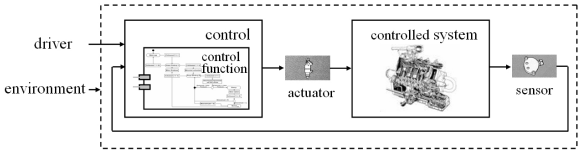


Figure 1: Closed loop control

Therefore, in this paper we present a tool chain for the development of families of control systems, exemplarily in the automotive domain. The tool chain

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

implements process family engineering techniques developed within the context of the PESOA [PES06] project[1]. We demonstrate the applicability of the tool chain by means of a windshield wiper case study.

This paper is structured as follows: in section 2 we outline the process family engineering process. This process serves as the conceptual basis for the tool chain described in section 3. The introduction of the tool chain is integrated with the description of the windshield wiper case study. In section 4 we give an overview of related work as well as an outlook to open issues.

2. Process Family Engineering

In this section we give an overview of the process family engineering process that serves as the conceptual basis for the tool chain described in section 3. For a description of the development activities supported by the tool chain, we refer to the classical six-pack model for product family engineering [Lin02]. However, in order to emphasize that we deal with families of process-oriented software we will use the terms process family engineering instead of product family engineering and process family infrastructure instead of product family infrastructure.
Figure **2** shows the corresponding development process.

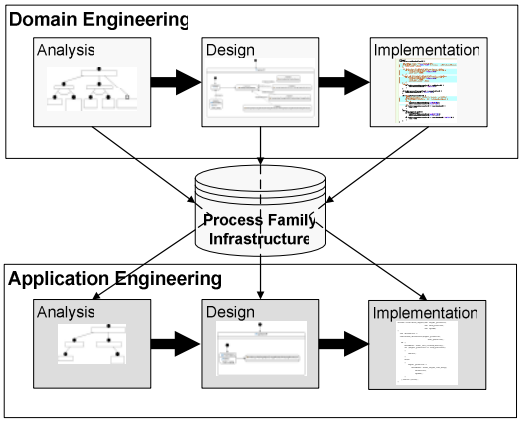


Figure 2: Process Family Engineering Activities

During the *domain analysis* activity we use generic feature diagrams to describe the requirements on the process family. Moreover, there can be additional analysis documents like for example descriptions of the external events and the expected reactions of the system. Such event lists are used in the case study described in section 3. In particular, the *domain analysis* activity implies the decision for a certain abstraction level. Real-time requirements comprising variabilities, for example, could be handled as elements of the process family, or, on a more technical level, completely be covered by the implementation activities.

Based on these requirements a process family architecture is developed during the *domain design* activity using a combination of variant-rich UML 2.0 State Machines and Activity diagrams [UML05]. According to the analysis in [BEL04] and the case studies described in [RSW04] a combination of UML State Machines and Activity Diagrams is most suitable for modeling software for embedded control units. For variability modeling, variant-rich process diagrams need to contain three additions to standard process diagrams [Sch06b]. The first addition is the identification of places where variability occurs (variation point). The second modification is that possible resolutions (variants) should be shown in the diagram. The third addition is the representation of the variability mechanism for realizing the variability.

In order to highlight variation points in UML State Machines we assign the stereotype «VarPoint» to the affected State Machine element. Variants, which are identified by the stereotype «Variant» are assigned to their variation point using UML Dependencies. The variability mechanism is modeled by assigning a stereotype with a name of the variability mechanism to the Dependency relation. Moreover, we suggest the introduction of the stereotype «Variable» to denote variability below the level of detail currently shown.

Most of the variabilities within the process family architecture can be linked to an arbitrary combination of features in the feature diagram. However, there are also variabilities, which are specific to the process family architecture. In order to allow for an automatic product derivation as well as for maintenance reasons, the dependencies between the variabilities in the feature model and the process family architecture have to be maintained. Moreover, the consideration of the variability dependencies assures that a consistent feature model and process architecture is derived during application engineering. These issues regarding variability tracing and variability dependency maintenance can be summarized under the term

[1] PESOA is a cooperative project partially supported by the German federal ministry of education and research (BMBF). (Förderkennzeichen: 01 ISC 34E). Its aim is the design and prototypical implementation of a process family engineering platform.

variability management. For the purpose of variability management, we introduce an additional model, the *variability management model*.

Within the process family architecture the optional tagged value *Preconditions* of the «Variant» stereotype can provide information about the dependency of the subprocess variant from a certain feature and variant configuration. Since these dependencies can become arbitrarily complex, we provide only a simplified dependency expression in the variant-rich process model. The full dependency expression is maintained by the variability management model. If the selection of a subprocess variant requires the selection of a number of features and variants, the *Preconditions* tagged value will contain these features and variants as a list of comma-separated names. However, if the dependency is more complex, e.g. if a certain subprocess variant shall only be selected if feature1 and feature2, and moreover variant2 have been selected, but not feature5 or feature6, the *Preconditions* tagged value would contain the following expression: F(feature1, feature2, variant2, feature5, feature6). This indicates that the dependency is a more complex one depending on the selection of the features and variants given as parameter values. Additionally, the stereotype «Variant» may contain the tagged value *Effects* providing information about the effects resulting from the selection of the variant. These are represented like the preconditions with the difference that no features, but only variants can be affected. More detailed information how to model variant-rich State Machines and Activity diagrams can be found in [Sch06a] and [Sch06c].

During the *domain implementation* activity a domain-specific generator and domain-specific components are developed. Based on the variant-rich processes, a generator is implemented supporting the automated creation of source code that performs the appropriate process variant according to a specified configuration (see *application implementation*). Generic functionalities, i.e. functionalities that are part of all product variants within the domain, are not covered by code generation. If such functionalities are, on the other hand, to be developed especially for the selected domain, they are implemented as domain-specific implementation components [BBG05], for example as libraries or frameworks that are specific for the selected domain. Typically, hardware-related routines are candidates for such components within the (superior) embedded domain. Second kind of domain-specific components are tools that are to be used during *application implementation*, but are developed specifically for the selected domain, e.g. special pre/postprocessors. These components are referred to as domain-specific infrastructure components [BBG05]. In our case study we will provide examples for both kinds of domain-specific components. The variant-rich UML process models, the feature model, as well as a prototypical product variant implementation provide the input data needed for the *domain implementation* activity.

During *application analysis* and *design* the specific requirements on the embedded system variant to be developed are captured based on the feature model and the variant-rich UML models. During product derivation, the dependencies and constraints within the variability management model have to be regarded. Any product configuration, being represented as a set of selected feature model and process model variants, has to fulfill them. This has to be proved during product derivation. For verification, the selected and deselected features and process model variants are represented as a set of expressions in propositional logic of the form *variant* = *true* (in case the variant has been selected) and *variant* = *false* (in case the variant has been deselected), while variant is the name of the literal representing the selected variant. So, in order to verify that a configuration is valid it has to be proved that the configuration fulfills the process family constraints. This can be done by means of a tool for checking the satisfiability of logical expressions (i.e. a SAT-checker). SAT-checker can also be used for calculating resulting configurations based on the process family constraints. The SAT-checker therefore checks whether new expressions of the form *variant* = *false* or *variant* = *true* can be deduced from the process family constraints and the current configuration.

During *application implementation* the product providing the functionality described in the application-specific process model is developed by applying the domain-specific generator. The generator imports the selected product configuration represented by the selected features and process variants and uses it to control the generation of target code. The generated target code is built and integrated considering the domain-specific components to the completed product. Finally, the resulting product is tested.

3. Tool Chain and Case Study

In this section we introduce a tool chain that implements the process family engineering concepts for

embedded systems from section 2. We also show how the tool is applied for the development of a family of windshield wiper control systems.

3.1 Domain Analysis

During the *domain analysis* activity a feature model and an event list were modeled. For managing and structuring features hierarchically, we have used pure::variants[2]. Within the feature model the commonalities and variabilities (optional, alternative, logical "or"-relations) among the features are documented. Moreover, it describes dependencies among the features using associations like "requires" and "conflicts". Figure 3 shows an extract of the feature model for the windshield wiper created with pure::variants.

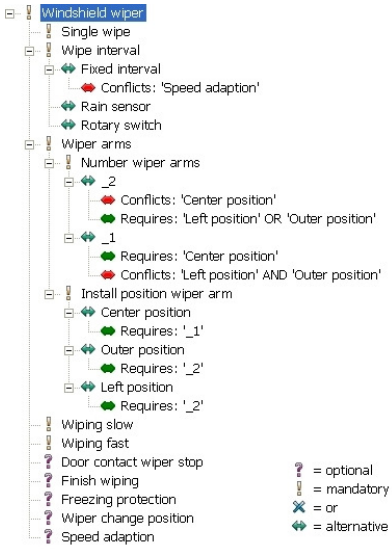


Figure 3: Extract of Feature Model "Windshield Wiper"

The event list documents the identified external events that trigger the windshield wiper. For the windshield wiper case study 25 external events were identified. The events were mapped to the elements in the feature model correspondingly. Figure 4 presents the documentation of the event "Ignition Off". It is activated by the user and impacts the windshield wiper.

No.	1
Event	Position ignition key turned on "off"
Response	Finish wiping
Classification	Control event / Data event
V(in)	Position ignition key, wiper speed, wiper position
V(out)	-
State(in)	Wiping process active
State(out)	Windshield wiper not ready
Precondition	Wiper not in parking position
Description	Only available with optional feature "Finish wiping"

Figure 4: Extract of Event List - Event "Ignition Off"

[2] Variant management tool from pure-systems GmbH

3.2 Domain Design

The feature model as well as the completed event list served as input for modeling the variant-rich process in the *domain design* activity.

For supporting the *domain design* activity the IBM Rational Software Architect [IBM06] has been enhanced by functionality for modeling process family variability. We call the resulting tool *Process Family Architect*. The Process Family Architect allows for modeling variant-rich UML State Machines and Activity diagrams following the concepts described in section 2. The functionality of the Process Family Architect includes the creation of variation points and variants. New variants are assigned to their variation points showing also the variability mechanism to be used for realizing the variability. In order to allow for a consistent and automized configuration of the variant-rich process models the Process Family Architect has been integrated with pure::variants. Thereby, it is possible to assign a variant in the variant-rich process model to a feature configuration within the feature model. Moreover, the Process Family Architect allows for assigning attributes to UML elements. These can be interlinked with the values of feature attributes in the feature model.

In order to create a new variation point using the Process Family Architect the UML element to become a variation point is selected. Next, the function *Make Variationpoint* is invoked from the menu or context menu as Figure 5 shows.

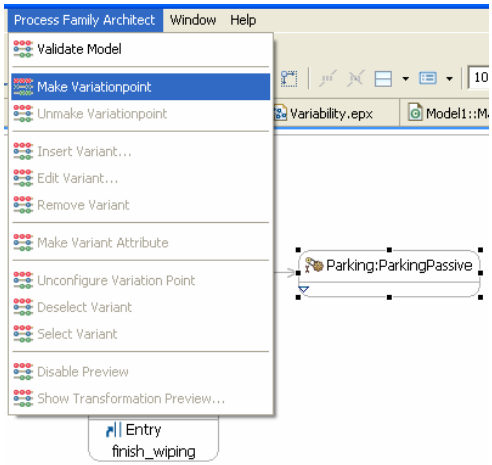


Figure 5: Process Family Architect Menus for Inserting Variation Points

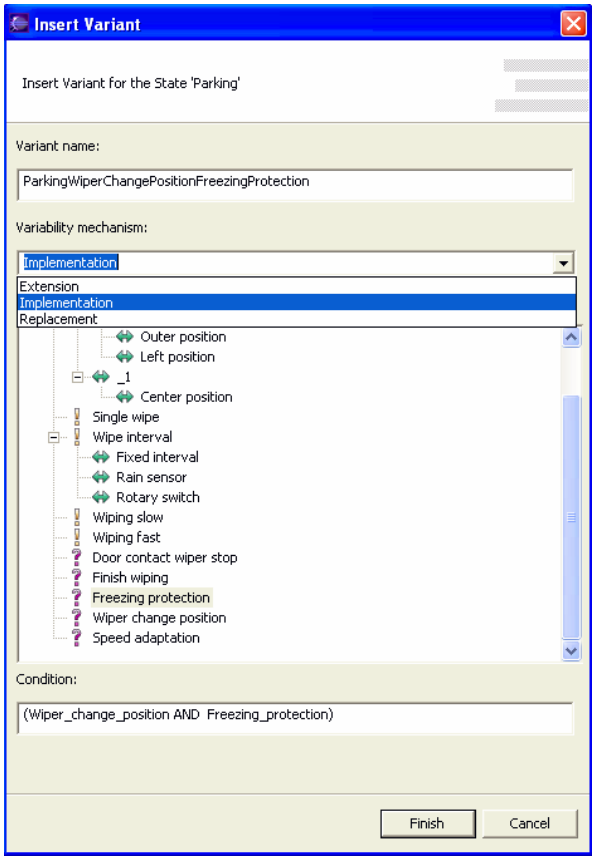


Figure 6: Process Family Architect Menu for Inserting Variants

After an element has been highlighted as variation point using the *«VarPoint»* stereotype, the new point *Insert Variant* becomes available for the variation point. For the new variant a variability mechanism has to be selected from a list. Which variability mechanisms are offered depends on the type of the variation point. Now, a selection condition can be described in propositional logic for the new variant. The variables for the selection condition can be selected directly from the feature model. The features can be interlinked using the operators AND, OR, and NOT. Figure 6 shows an example for the definition of a new variant for the variation point created in Figure 5. In this case, the variability mechanisms extension, implementation, and replacement can be selected since the variation point is a submachine state. Moreover, the selection of the new variant during configuration depends on the selection of the features "Wiper change position" and "Freezing protection" as the selection condition indicates.

The selection conditions stored in the feature model and variant-rich process model are part of the variability management model mentioned in section 2.

Extracts of the resulting variant-rich process model created by means of the Process Family Architect are shown in Figure 7 and Figure 8. The depicted process part specifies the behavior of the system when the event "IgnitionOff" occurs, i.e. when the system switches over to the state "IgnitionOff".

The variant-rich State Machine in Figure 7 contains two submachine states acting as variation points as indicated by the «VarPoint» stereotype. The variability on the left-hand side is realized by means of the variability mechanism "Extension". One variant is a null-submachine without behavior. The other process variant is a behavioral extension which is represented by the encapsulating submachine "Finish Wiping".

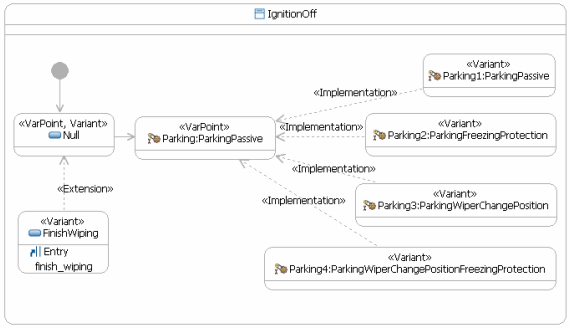


Figure 7: Variant-rich process - UML State Machine diagram "IgnitionOff"

Which variant is selected depends on whether the feature "Finish Wiping" has been selected for the windshield wiper or not. For realizing the variability on the right-hand side the variability mechanism "Implementation" was used. The variants which were connected to the variation point were specific submachines that could be invoked alternatively by the submachine state acting as variation point.

An example for a variant-rich Activity diagram is shown in Figure 8. It describes the behavior of the system in the state "FinishWiping" (see Figure 7). In case of "FinishWiping" the wiper is brought to an end position when the ignition is turned off. For the variant "Door Contact Wiper Stop" the behavior is extended by the functionality that wiping is stopped temporarily in case an optional door sensor indicates that the driver has opened the door.

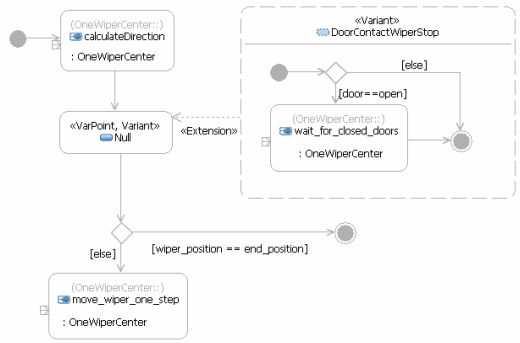


Figure 8: Variant-rich Process - UML Activity diagram "Move Wiper"

The variability mechanisms "Extension" and "Implementation" are special cases of the variability mechanism "Encapsulation of varying sub processes". Encapsulation of varying sub processes supports the restriction of variability to certain well-defined areas within the systems. Thereby, the maintenance and extensibility of the process family is supported. However, in contrast to other variability mechanisms, it may restrict the optimal reuse of common system parts.

3.3 Domain Implementation

The *domain implementation* work started with separating generic functionalities from functionalities to be covered by the code generator. In this example, hardware-related functionalities were necessary to provide basic access to a motor, a rain sensor, etc. These functionalities are specific for our selected domain, but neither related to any variation points within the variant-rich process models, nor to the features of the feature model. Therefore, they could be implemented as domain-specific implementation components. In our case, we defined a "hardware abstraction layer" comprising all hardware-related routines. In detail, this domain-specific component was represented by the C++ classes[3] "Rainsensor", "Doorcontact" and "Motor".

The other functionalities were to be covered by the domain-specific generator. For its development the HyperSenses[4] technology was used. It provides a convenient model-based approach for the implementation of generators. The concrete generator development tool is named *HyperSenses Meta Composer* [HS06]. A HyperSenses generator consists of so-called "code patterns" and a metamodel following the OMG MOF standard [MOF06]. The metamodel comprises all variation points that are to be considered. The code patterns are parameterized fragments of target code, which are coupled to the metamodel. The generator had to be capable of processing the selected features and process variants issued by the Process Family Architect. Thereby, the generator considered the variation points of the process as variation points for code generation. As the OMG XMI standard [XMI05] is defined especially to connect modeling tools with code generators, it was chosen as input format. Additionally, the generator had to be configured for the appropriate XMI data, as XMI is a standard allowing different XML schemas. In our case the generator received its input from an XMI file generated by the Process Family Architect. This XMI file contained the product configuration, i.e. the selected features and process variants.

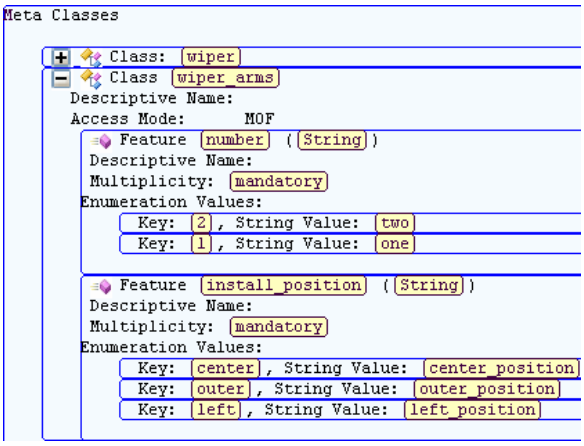


Figure 9: HyperSenses Metamodel Section with MOF Class Definition "wiper_arms".

The development of the domain-specific generator using HyperSenses started with defining a metamodel (see Figure 9).

This metamodel was defined using MOF elements describing the variation points of the process. The metamodel was defined interactively, using the HyperSenses Meta Composer. The MOF features depicted in Figure 9 and Figure 10 correspond to features of the feature model that have no sub-features, while the MOF classes are used to construct a hierarchy, according to superordinated features of the feature model. Mandatory features with fixed values are, however, not considered in this metamodel: our

[3] Usually, software in the automotive domain is implemented in the programming language C. Our environment and controller allows us to implement the software in C++. The concepts described here can also be applied to C.

[4] HyperSenses™ is a trademark of Delta Software Technology.

metamodel comprises exactly the variation points needed for the generation task.

```
Feature freezing_protection (Boolean)
Descriptive Name:
Multiplicity: optional
Feature wiper_change_position (Boolean)
Descriptive Name:
Multiplicity: optional
```

Figure 10: HyperSenses Metamodel Section with MOF Features ”freezing_protection” and ”wiper_change_position”.

Based on the metamodel as the main artifact of the generator's implementation, so-called ”code patterns“ were defined. Within HyperSenses, code patterns are profoundly parameterized code fragments, coupled to elements of the metamodel [HS06]. In our domain, for example, the ”Wiper“ class contains a method to control the first wiper motor, i.e. the only one or the left one (see Figure 11). The corresponding part of the code pattern comprises several ”slots“ (dark grey / magenta) and “optional code blocks“ (light grey / turquoise).

Slots are directly associated to metamodel features, receiving string values or values processed by expressions. In our case, slots were especially useful to implement variable moving directions of the wiper. Consider the slot definitions for “moveLeftB“ and ”moveRightB“, as shown in Figure 12. The corresponding expression is a ternary if-condition which checks for the install position of the first wiper. If this wiper is installed in the ”center“ position, it starts from the left, wiping to the right. If two wipers are installed, the first one is installed on the left side in any case (i.e. ”left“ and ”outer“). In this case, it starts from the right moving to the left.

Optional code blocks define whole code sections as being coherent code generation elements. They are integrated or left out in the target code only as a whole. Each optional code block depends on embedded slots – they all have to be filled with values to enable the generation of the block. Additionally, an optional code block may have a condition [HS06]. Such a condition consists of an expression yielding a true or false value. In our case, each block was defined with one condition, as shown in Figure 12: The elements with document icons represent optional code blocks. Most of them depend on the features ”wiper_change_position“ or ”freezing_protection“. Code blocks actually may be nested, whereby the inner block depends on the outer one. An example is the block named ”doNothing Block“ highlighted in Figure 12, which comprises the three nested blocks ”info_changePosition“, ”or_operator“ and ”info_icePosition”.

```
//Motor1
void Wiper::sendImpulsePortB() {

    //continue wiping after interval time
    if (info.Function == cClearance && counter > info.currentClearance)
        info.pause = FALSE;
    }

    if (info.changePosition) { //wiper change position
        if (info.currentPositionPortB == changePosPortB) {
            info.directionPortB = moveLeftB;
            info.pause = TRUE;
        }
    }
    else if (info.icePosition) { //freezing protection
        if (info.currentPositionPortB == icePosPortB) {
            info.directionPortB = moveLeftB;
            info.pause = TRUE;
            info.icePosition = FALSE;
        }
    }
    else {
        if (info.currentPositionPortB == endPosPortB) {
            info.directionPortB = moveLeftB;
        }
    }

    if (info.currentPositionPortB == startPosPortB) {
        info.directionPortB = moveRightB;
    }

    if ( ! info.directionPortB) {
        //if wiper change position or freezing protection
        //are doing nothing:
        if ( info.changePosition || info.icePosition ) {
        }
        else {
            motor->moveEngineOneStep( portBLeft );
            info.currentPositionPortB decrementB;
        }
        if (info.currentPositionPortB == startPosPortB) {
            cout << 0;
        }
    }
    else {
        //if pause do nothing
        if (!info.pause) {
            motor->moveEngineOneStep( portBRight );
            info.currentPositionPortB incrementB;
        }
        if (info.currentPositionPortB == startPosPortB operator 2) {
            cout << 2;
        }
    }
}
```

Figure 11: Code Pattern Source Definition Section Containing Several Slots and Optional Code Blocks.

While slots are used to associate target code positions with values, optional code blocks provide a declarative mechanism to express even very complex dependencies between several fragments of target code.

In fact, the definition of metamodel elements and patterns was an interactive process, whereby the sequence of metamodel and pattern definition remarks the first working cycle. After finishing this work, the generator implementation had to be completed by the model import [GOB05, chapter 4]. Such a model is an instance of the metamodel, following the definition and philosophy of MOF. A HyperSenses model therefore comprises concrete class instances and feature values, representing the configuration data needed for code generation. As stated above, these data was available in XMI format. The model import had to be able to process these XMI data. Therefore, it was configured by mapping definitions, considering the specific XMI format as well as the specific metamodel. In particular,

such mapping definitions are true domain-specific infrastructure components (see chapter 2).

Used Slots and Blocks	
clearance	interval = "fixed" or interval = "rain"
cClearance	interval = "rain" ? "cClearanceRainSensor" : "cClearance"
change_position	wiper_change_position = "on"
moveLeftB	wiper_arms_install_position = "center" ? "moveLeft" : "moveRight"
else1	wiper_change_position = "on" and freezing_protection = "on"
ice_position	freezing_protection = "on"
moveLeftB	wiper_arms_install_position = "center" ? "moveLeft" : "moveRight"
else	wiper_change_position = "on" or freezing_protection = "on"
moveLeftB	wiper_arms_install_position = "center" ? "moveLeft" : "moveRight"
moveRightB	wiper_arms_install_position = "center" ? "moveRight" : "moveLeft"
not	wiper_arms_install_position = "center"
doNothingBlock	wiper_change_position = "on" or freezing_protection = "on"
info_changePosition	wiper_change_position = "on"
or_operator	freezing_protection = "on" and wiper_change_position = "on"
info_icePosition	freezing_protection = "on"
portBLeft	wiper_arms_install_position = "center" ? "portFLeft" : "portBRight"
decrementB	wiper_arms_install_position = "center" ? "--" : "++"
doNothingBlockClose	wiper_change_position = "on" or freezing_protection = "on"
portBRight	wiper_arms_install_position = "center" ? "portFRight" : "portBLeft"
incrementB	wiper_arms_install_position = "center" ? "++" : "--"
operator	wiper_arms_install_position = "center" ? "+" : "-"

Figure 12: Code Pattern Properties Containing Code Block Conditions.

3.4 Application Analysis

In *application analysis* the generic feature model is instantiated regarding the requirements of a specific product. For this purpose we select those features out of the windshield wiper feature model which lead to a valid feature configuration. In order to accomplish this process step we used the tool pure::variants (Figure 13).

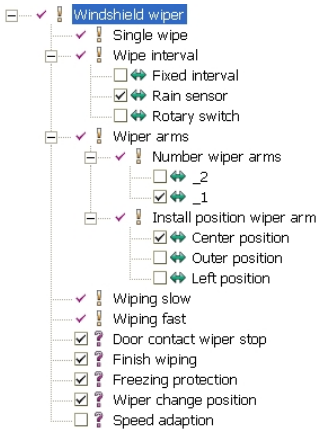


Figure 13: Feature Selection out of the Windshield Wiper Feature Diagram

3.5 Application Design

During the *application design* activity the variant-rich UML State Machine and Activity diagram models are now configured automatically within the Process Family Architect regarding the selected features. The automatic configuration of the variation points in the variant-rich UML diagrams is based on an evaluation of the selection conditions for every variant. The value of the selection conditions depends on the feature configuration selected in pure::variants.

Figure 14 and Figure 15 show the configured UML State Machine diagram "Ignition Off" and the configured UML Activity diagram "Move Wiper".

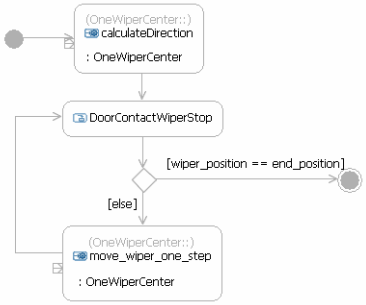


Figure 14: Specific Process Model - UML Activity Diagram "Move Wiper"

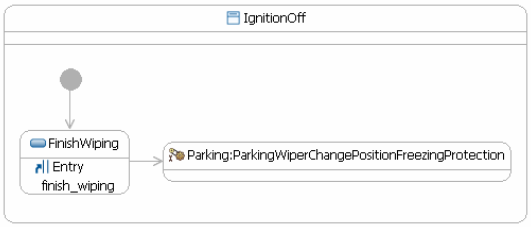


Figure 15: Specific Process Model - UML State Machine Diagram "IgnitionOff"

3.6 Application Implementation

The *application implementation* work started with importing the product configuration data – the selected features and process variants. These data have been issued by the Process Family Architect as an OMG XMI file. The file is imported and transformed by HyperSenses' model import into a HyperSenses model [GOB05, chapter 4], an instance of the metamodel defined at *domain implementation*. This model was already complete, that is, the code patterns could immediately be applied to it. The model import and code generation tasks were triggered from the tool *HyperSenses Active Intent*.

In HyperSenses, code patterns may form hierarchies, whereby one such hierarchy forms one, so-called "rendering". In general HyperSenses renderings are

used either for configuration (model creation) purposes or for code generation. On a technical level, there is actually no difference: any rendering provides a certain view on the model data. As an important consequence, a code generation is triggered by just switching to the appropriate rendering, compare the concepts outlined in [CE00, chapter 11]. One rendering thereby is associated to one output target code file [HS06]. In our case, there was a range of files to be generated. Therefore, HyperSenses was extended by a script that automatically starts all possible code generations for one model.

The generated C++ files were put together with the hardware abstraction layer sources and built within a Cygwin environment on a Windows notebook. The target platform consisted of a microcontroller and a rain sensor as well as a door open sensor. The controller was an AVR128 8-Bit RISC processor with 128 KB data flash and 4 KB SRAM. This hardware environment required certain code optimizations, for example, inline method definitions. They were needed to reduce the stack size, which was limited by SRAM size. For the controller the PURE operating system was used [SSB03]. The mentioned code optimizations took place within the code pattern definitions, leading to a short development cycle between *domain implementation* and *application implementation*. Finally, the whole executable code was flashed from the notebook to the controller to become tested.

4. Conclusions

In this paper we have presented a tool chain for developing families of embedded systems.

For developing product families a number of alternative approaches have been proposed, e.g. in [ABB02, KCH90, PBL05, PRB03, WeL99]. However, these are rather targeted towards structural diagrams, while variant-rich behavioral diagrams are not in the center of interest. Especially, for embedded control software – as part of embedded control processes – modeling dynamic aspects is indispensable. On the other hand those approaches focusing on variant-rich processes are not integrated into a process family engineering development process [KLB05, McS03, Pre04, RoA03]. Moreover, they do not completely fulfill the requirements on variability modeling in process family architectures as analyzed in [Sch06b] and focused within this paper.

Based on the case study of a windshield wiper control, we have shown the applicability of the process family engineering approach. However, our approach is not limited to automotive control processes but can be applied to the development of arbitrary families of embedded control software. The only prerequisite is that the system can be described by means of UML. This holds especially for those domains, where modeling of system behavior is of special interest, for example in the telematic domain. Finally, we have shown the feasibility of our approach by means of an integrated tool chain.

Modeling and configuring variant-rich processes are per se complex tasks. With this knowledge in mind, it has to be evaluated how far the tool chain also scales for larger systems. In this context reuse of already existing model artefacts plays an important role. An open issue for future work is integrating reengineering aspects into the process family engineering and the adaption of tool support, respectively. Here, we consider aspects for seamlessly integrating variability into already existing architecture models.

5. References

[ABB02] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Paech, Jürgen Wüst, and Jörg Zettel. Component-based Product Line Engineering with UML. The Component Software Series. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2002.

[BBG05] J. Bayer, W. Buhl, C. Giese, T. Lehner, A. Ocampo, F. Puhlmann, E. Richter, A. Schnieders, J. Weiland. Process Family Engineering. PESOA Report No. 18/2005, August 2005, available on http://www.PESOA.org

[BEL04] J. Bayer, M. Eisenbarth, T. Lehner, F. Puhlmann, E. Richter, A. Schnieders, J. Weiland. Domain Engineering Techniques and Process Modeling. PESOA-Report No. 09/2004, DaimlerChrysler Research and Technology, Fraunhofer IESE, Hasso-Plattner-Institut, Oktober 2004.

[CE00] K. Czarnecki, U. Eisenecker. Generative Programming – Methods, Tools, and Applications. Addison-Wesley, Boston, MA, 2000

[GOB05] C. Giese, H. Overdick, W. Buhl. Realisierungsstrategien für Prozessfamilien (Realization Strategies for Process Families; in German). PESOA Report No. 15/2005, August 2005, available on http://www.PESOA.org

[HS06] Delta Software Technology GmbH. HyperSenses Tutorial. May 2006, available on http://www.D-S-T-G.com/HS

[IBM06] IBM Rational Software Architect. Homepage: http://www-306.ibm.com/software/awdtools/architect/swarchitect/. (last accessed in November 2006)

[KCH90] Kyo Kang, Sholom Cohen, James A. Hess, William E. Novak, and A. Spencer Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, 1990.

[KLB05] D. Karastoyanova, F. Leymann, and A. P. Buchmann. An Approach to Parameterizing Web Service Flows. In ICSOC, pages 533–538, 2005.

[Lin02] Frank van der Linden. Engineering Software Architectures, Processes and Platforms for System Families - ESAPS Overview. In Proceedings of the Second International Conference on Software Product Lines, Lecture Notes In Computer Science; Vol. 2379, pages 383 – 398, London, UK, 2002. Springer-Verlag.

[McS03] A. T. McNeile and N. Simons. State Machines as Mixins. Journal of Object Technology, 2(6):85–101, 2003.

[MOF06] Object Management Group. Meta Object Facility (MOF) Core Specification, Version 2.0. January 2006.

[PES06] PESOA Projekt. Homepage: http://www.PESOA.org. (last accessed in November 2006)

[PBL05] Klaus Pohl, Günter Böckle, and Frank van der Linden. Software Product Line Engineering: Foundations, Principles, and Techniques. Springer, 2005.

[PRB03] I. Philippow, M. Riebisch, and K. Boellert. The Hyper/UML Approach for Feature Based Software Design. In Proceedings of 4th AOSD Modeling With UML Workshop, 2003.

[Pre04] C. Prehofer. Plug-and-play Composition of Features and Feature Interactions with Statechart Diagrams. Software and System Modeling, 3(3):221–234, 2004.

[RoA03] M. Rosemann and W. van der Aalst. A Configurable Reference Modelling Language. Technical Report QUT Technical report, FIT-TR-2003-05, Queensland University of Technology, Brisbane, 2003.

[RSW04] E. Richter, A. Schnieders, and J. Weiland. Prozessanalyse und Modellierung in der Automotive Domäne (Process Analysis and Process Modeling in the Automotive Domain; in German). DaimlerChrysler Research and Technology, Hasso-Plattner-Institute for IT Systems Engineering, Tech. Rep., Oktober 2004.

[Sch06a] A. Schnieders. Modeling and Implementing Variability in State Machine Based Process Family Architectures for Automotive Systems. 3rd International Workshop on Software Engineering for Automotive Systems - SEAS 2006. In Proceedings of the 28th international Conference on Software Engineering (Shanghai, China, May 20 - 28, 2006). ICSE '06. ACM Press, New York, NY, 1034-1034.

[Sch06b] A. Schnieders. On the Architectural Relevance of Variability Mechanisms in Product Family Engineering. In Proceedings of the Workshop on Managing Variability for Software Product Lines: Working With Variability Mechanisms, in conjunction with the 10th International Software Product Line Conference (SPLC 2006), Baltimore, 2006.

[Sch06c] A. Schnieders. Variability Mechanism Centric Process Family Architectures. In Proceedings of the 13th Annual IEEE International Conference on the Engineering of Computer Based Systems ECBS 2006, pp. 289-298, IEEE Computer Society Press, 2006.

[ScZ03] Jörg Schäuffele, Thomas Zurawka. Automotive Software Engineering – Grundlagen, Prozesse, Methoden und Werkzeuge (Automotive Software Engineering – Basics, Processes, Methods, and Tools; in German). Vieweg, Juli 2003.

[SSB03] O. Spinczyk, W. Schröder-Preikschat, D. Beuche, H. Papajewski. PURE/OSEK – Eine aspektorientierte Betriebssystemfamilie für Kraftfahrzeuge (PURE/OSEK – An Aspect-Oriented Operation System Family for Automobiles; in German). In Informatik 2003 – Innovative Informatikanwendungen, Kölln Druck+Verlag, Bonn, 2003.

[UML05] OMG. UML 2.0 Superstructure Specification, August 2005.

[WeL99] David M. Weiss and Chi Tau Robert Lai. Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley, 1999.

[XMI05] Object Management Group. XML Metadata Interchange (XMI) Specification Version 2.0. May 2005.

A Look at Typical Difficulties in Practical Software Development from the Developer Perspective
A Field Study and a First Solution Proposal with UPEX

Ivonne Erfurth
Friedrich-Schiller-University Jena
07743 Jena, Germany
ivonne.erfurth@inf.uni-jena.de

Wilhelm R. Rossak
Friedrich-Schiller-University Jena
07743 Jena, Germany
rossak@inf.uni-jena.de

Abstract

In the first part of this paper we summarize the results of a questionnaire for software professionals which we, the Software Engineering Group at Friedrich-Schiller-University Jena, handed out to (local) small and medium-sized IT businesses. The questionnaire targeted the true state of the art in practical software development with a focus on questions regarding the benefits and drawbacks of standard methodologies and all forms of user participation in such an environment. Especially interesting for us was the degree of customer integration into the development process, the understandability of used notations from the developer and user perspective, and the possible reasons for difficulties in the interaction between customer and developer teams.

In the second part we present a first look at the UPEX methodology which takes into account the results of the questionnaire.

1 Introduction

As many studies [8, 9, 5] show, a majority of all projects miss their goals. This percentage is presented by the Standish Group [6] which publishes research reports on the success of software projects in regular intervals. Starting with the first release 1994 [7], the report indicates every single time that about 70 percent of all projects miss their goals.

These reports, as many others, indicate a clear problem, a recurring symptom for nearly all branches of the software production domain. For us, the interesting question must be what the reasons and attributes are that trigger on a regular basis situations where too many project miss their goals.

In this context it is important to understand that, in connection with the modeling and specification of complex software systems, not only technical problems cause projects to fail. Most abstract and formal models, which provide a concise and clean basis for the developer, are not easy to be understood for the stakeholder and, thus, lead to costly misunderstandings already in the early phases of systems development. Just a remark, stakeholder means both customer and future user of a software system. We targeted this situation in the upstream phases of software development an tried to pinpoint what the true difficulties between stakeholder and developer are. To this end we moved into the IT-based industrial environment here in Thuringia and developed an online questionnaire that targets the typical local, small and medium-sized business.

Goal of the questionnaire was to get answers regarding the state of art in practice. Which process models, methods and notations are really used? When, where and how arise problems between stakeholder and developer teams during the development process? In addition it was important for us to get an answer how far and in which way the stakeholder is integrated in the development process. We received 65 answered questionnaires from 54 companies, mostly with less than 50 employees.

In this paper we describe structure and content of the questionnaire. An evaluation will then provide some insight in the local situation. While we are aware that the questionnaire is not fully representative in selection and quantity of the sample, we still believe that it gives us a chance to have a deeper look into true development practices for this type of companies. It will also help us to place our future research close the practical needs in our local environment and open an opportunity for targeted improvement which drives us to a first solution proposal called UPEX.

2 Structure of our questionnaire

Our questionnaire was structured into five parts (see figure 1).

Part I. and II. helped to clarify general questions, as size

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

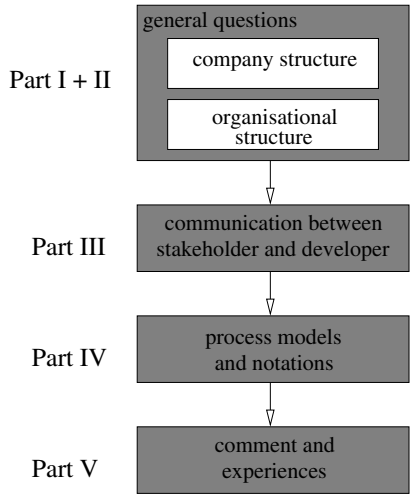


Figure 1. Structure of our questionnaire

of business and project teams, the volume of projects in person-month, as well as the type of software developed in the enterprise. The qualifications of interviewers and stakeholders had to be determined, too. The III. part of the questionnaire researched the communication between stakeholder and developer team, as to how and when the stakeholder is usually integrated in software development. The IV. part asked for process models and notations. It also inquired for an estimate how well understood the used notations might be from the stakeholder perspective. The last part asked for specific and typical difficulties that appear on a regular basis and how the enterprise tries to avoid them currently.

3 Evaluation of the questionnaire

In most cases the interviewee was a software developer. Thus, our results reflect the opinion of this group of professionals. We plan a second questionnaire where we intend to target the stakeholder population to cross-reference the so far achieved results.

The evaluation is divided into four parts which are mainly derived from the main goals of the questionnaire. Fist we look at the characteristics of the involved enterprises. Then we describe the state of art in practice, the problems between stakeholder and developer teams, and, finally, the achieved level of direct stakeholder integration.

3.1 Characteristics of interviewed businesses

75 percent of the asked businesses have less than 50 staff members. So we speak of small and medium-sized enterprises. Most of the companies develop customized software (over 50 %), but they also adapt standard software to their customer's needs (Figure 2).

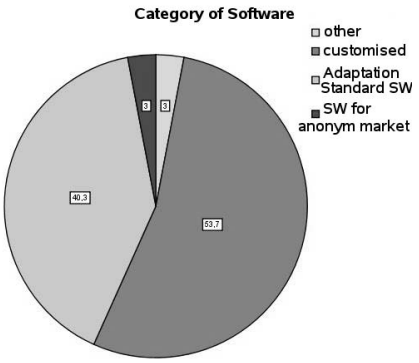


Figure 2. Category of software

The size of development teams in the software projects is in most cases less than 5 members (answered 75 %). Teams with more than 10 members are seldom. Thereby, it makes no difference, if we speak about a small enterprise or an enterprise with more than 100 employees.

The duration of projects is seldom over 24 man-month. In about 50 percent of all cases, the projects are implemented in less than 12 person-month and in about 40 percent, the projects duration is between 12 and 24 person-month.

Most interviewees were developers (over 60 %), but we also reached controlling and management (25 %) as well as sales and marketing personal (10 %). The qualification of stakeholders, which are integrated in software projects, are half-and-half business management and technical personal.

3.2 State of the art in practice

In response to the question "Which process models, methods and notations will be used?" (multiple choice possible), only 55 percent of the interviewed companies answered to use a structured approach. At first sight this low percentage seemed alarming. But at a second view it turned out that over 70 percent were using process models when the duration of a project is longer than 12 man-month. Most enterprises (about 45 %) use prototyping. Each third company uses incremental models. Other approaches like the waterfall model, spiral model, V-model, or Unified Process are used by about 15 percent only (see figure 3).

The more members in the team the more often the waterfall model is used. Independent from the duration of the project different methodologies will be used. Conspicuously is that prototyping tends to result in more communication time with the stakeholder.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

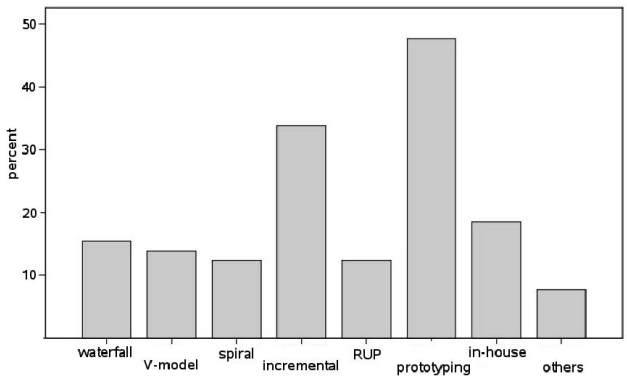


Figure 3. Percentage of used process models

Regarding used methods and notations, mostly object oriented methods are common. In most cases class diagrams are used, followed by use case diagrams and sequence diagrams (see figure 4)

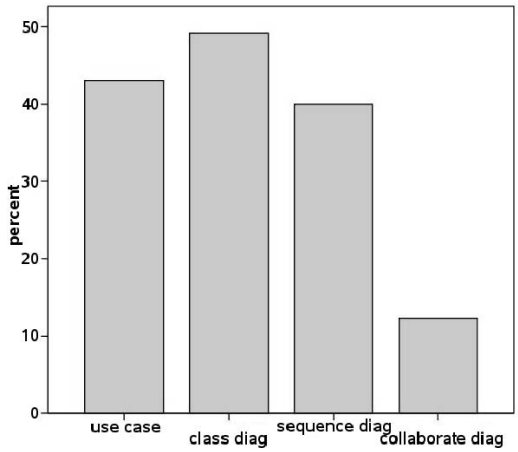


Figure 4. Percentage of used object oriented notations

Non object oriented methods are used less frequently. In addition to object oriented methods ERDs (Entity Relationship Diagrams) are used often (about 30 %). Work flows and State Charts are seldom used. Companies which are working with object oriented methods are using different notations in parallel, because of the variety offered by the underlying UML. Non object oriented methods like ERD are used independently from other methods.

A good tool support, understandability and easy learning for developer and employees, as well as familiarity with the notation have been denoted as motives for the use of object oriented methods.

Next, we looked at the notations used to communicate directly with stakeholders. In spite of a high percentage for the use of class diagrams during system development, it is used to communicate with stakeholder only in about 10 percent of all cases. If companies work directly with the stakeholder, they tend to present use case diagrams.

Over 60 percent of our interviewees have the opinion that object oriented methods are in general not understandable for stakeholders (We have had a four-step classification and 50 % said: difficult to understand; 13 % said: not understandable). The same holds for class diagrams (53 %: difficult to understand; 20 %: not understandable). In contrast, 40 percent of the interviewees had answered that a use case diagram is an understandable notation.

It also seems that the notation presented to the stakeholder depends on how the stakeholder's qualification (see figure 5) is judged by the developer. If the stakeholder is business management personal, then developers also present work flows. If the stakeholder is technical personal, they will present use case diagrams and also class diagrams, state charts and sometimes formal methods like petri nets.

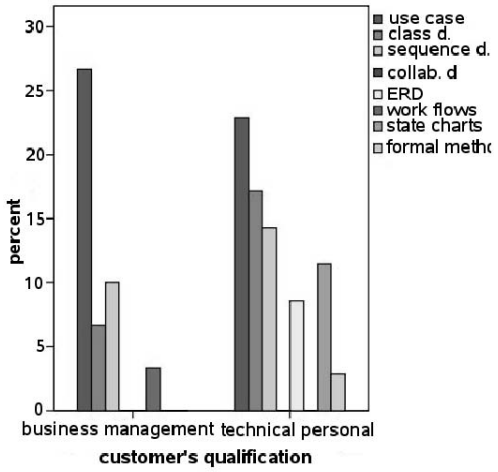


Figure 5. Used models depending on the stakeholder's qualification

3.3 Levels of stakeholder integration

A certain tendency that it could be beneficial to integrate not only policy makers into the development process can be found in most cases. Over 80 percent of the questioned companies communicate also with policy makers and future end-users. Only 10 percent solely communicate with policy makers.

Pro rata to the overall development process communication time with the stakeholder is between 10 and 30 percent. 20 percent estimate the communication time to be under 10

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

percent. 25 percent stated that direct communication time is over 30 percent within the overall development process.

During the requirements (Req.) phase, the integration of stakeholders is over 50 percent. In the modeling phase (Mod.), test phase (Test) and installing phase (Inst.) the stakeholder's integration is between 10 and 30 percent. In the implementation phase (Impl.), as expected, under 10 percent. See figure 6.

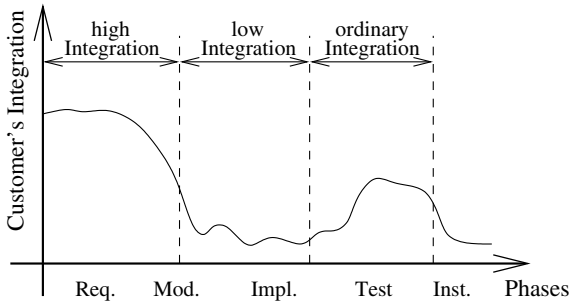


Figure 6. Stakeholder's integration into development process

Small and medium-sized enterprises tend to have a higher communication rate with the stakeholder. The smaller the project teams, the higher the communication percentage. The longer the project's duration, the lower the communication rate. Interestingly enough, communication efforts peak during adaptation of standard software.

To get more information regarding the understandability of notations, we looked at companies with a higher communication rate during the modeling phase. Enterprises which communicate with the stakeholder in the modeling phase at a rate of more than 30 percent also use class diagrams more often (about 60 %).

3.4 Problems between stakeholders and developers in practice

We asked directly what problems typically arise between stakeholder and developer teams during the development process. The results indicate that in the modeling, implementation and test phase significant problems do not occur or can be avoided. However, in the requirements phase problems occur in most cases: 30 percent of the interviewees answered that problems occur sometimes, while 30 percent said that problems occur frequently.

As one important conclusion of the questionnaire we got a ranking of possible difficulties during the development process (see Figure 7):

(1) imprecise customer requirements
(2) changes in requirements

but also

(4) misunderstandings because of different comprehension of terms
(5) stakeholders are not familiar with common techniques in software engineering and
(6) developers are not familiar with customer's domain problems.

These factors cause major problems during the software development process. Figure 7 depicts the actual percentage of participants who selected the answers from predetermined choices in the questionnaire. Selection number 3 (company policy) was marked at a non significant level.

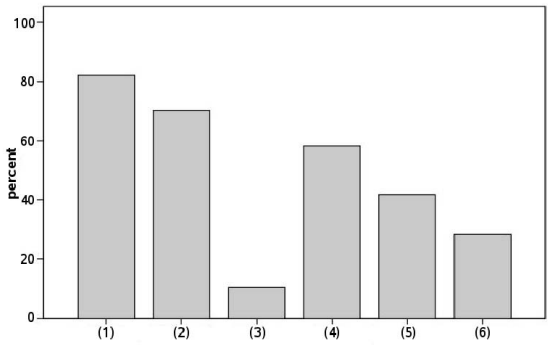


Figure 7. Reasons for deficiency in percent

Enterprises who indicate problems during the development process also declare that in most cases imprecise customer needs, unclear definition of requirements and ambiguities during the development process cause these difficulties.

4 Conclusions and a first approach to avoid problems – UPEX

4.1 The UPEX concept

As studies indicate clearly, there are difficulties to finish software development projects successfully in an industrial environment. We could show that the failure of projects is often triggered by difficulties in the direct and indirect communication between developer and stakeholder teams. For the stakeholder it is hard to understand frequently used terms, process models, and technological concepts. Developers have a hard time to understand domain specific processes and structures, and exhibit a tendency to abstract concrete examples to higher level constructs.

However, the questionnaire has also indicated that there is a growing awareness for the need to increase and improve

stakeholder participation in the future (at least in our local IT environment of SMEs).

A key concept in this context must be the use of notations that are adequate for a direct presentation to the stakeholder. As we believe, there is a need to elicitate, explain and discuss the possibly complex functionality of a system using instance based models, especially in the case of the development of complex distributed systems. This is a feature neglected in most current methodologies which strive to reach the level of classes and types as fast as possible, loosing the stakeholder on the way to an elegant model that is developer focused.

Hence, we currently develop a new customer oriented referencemodel, named UPEX [1] (User Participation by EXample) to gather and present the stakeholder's expertise. The general idea of UPEX is to enable a flexible interplay between requirement analysis and system design. Furthermore, we try to mediate in conflicts between stakeholders and developer during the software development process by providing on the one hand a presentation of static elements on the level of the stakeholders real-world entities. On the other hand we simulate the dynamic aspects of complex systems without immediate abstraction from examples.

For an evaluation of UPEX from the end-user perspective we work together with farmers [2]. This type of stakeholder is specially qualified for the evaluation of UPEX, because of his / her high expertise in the complex agricultural domain and low expertise in the area of computer science. Furthermore, it is very difficult to gather the expertise of farmers, because it is mostly tacit knowledge. Tacit knowledge means implicit and subjective knowledge. The farmer knows it, but does not talk about it. That makes it difficult for a non agriculture specialist to identify a work flow in an agriculture working process. Often during the development of a software system, developers have to identify processes of different domains in which the developer is not an expert.

How can we solve such problems? As the questionnaire shows, it is not adequate to present formal and abstract models of software engineering to farmers. Hence, we are looking for easy understandable methods to get tacit knowledge of different domain experts? This drives us to the general approach of UPEX (see figure 8).

In a first step the developer gets, for example, initial information / examples from farmers in interviews or other information collecting techniques of system engineering. These examples are mostly non structured and incomplete information. Misunderstandings and varied interpretations are preassigned and aberration tends to bring projects to fail. With the help of UPEX the analyst / system designer prepares this information structured for instance in an understandable semiformal work flow by using real-world examples and entities of stakeholders. In a second step developers and stakeholders analyze the process (work flow) in more detail to identify general rules.

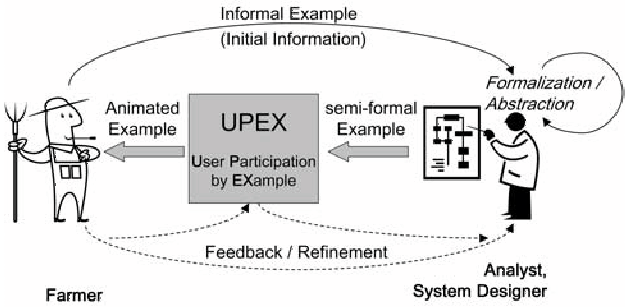


Figure 8. UPEX – User Participation by EXample

4.2 The UPEX structure

All in all our research identifies four main parts of UPEX (see also [1]):

1. Representation Layer
2. Static Entity-Structures (by example)
3. Work Flow-Structures (by example)
4. Visualization/Animation Layer

Representation Layer In software development prototyping is a more and more common technique. Hence, our questionnaire shows about 45 percent of interviewed enterprises use prototyping. An advantage of prototyping is an early presentation of a software solution to the customer. The customer participates in the software development process and decides on various aspects of his own software. However, prototyping bears some problems. Especially during the development of complex systems a chosen architecture may be wrong. If the worst comes to the worst the development has to be aborted.

We limit prototyping to techniques for designing a user interface. Interesting is the technique of paper prototypes, like PICTIVE by Muller [4]. Screens, menus, buttons and messages can be created using common office material like colored paper, markers, cards, pens etc. Every screen shot can be designed together with the stakeholders by using common office materials. Due to rising acceptance of computer usage this kind of prototyping can be computer-aided performed with an "online paper prototype". Digitalized office material can be displayed on a computer to construct a simple version of a user interface in a hands-on manner by future users. Stakeholders can recolor screens, move buttons and create menus until the interface fits their vision and needs.

During the design of the user interface the developer gets also hints of new entities and work flows of the stakeholders domain and new static and dynamic elements can be identified.

Static Entity-Structures In this part of UPEX the static elements of the system will be modelled. Looking at the results of our questionnaire in most cases an abstraction is not understandable for non computer scientists. For example regarding the understandability of class diagrams, 53 percent of the interviewees said class diagrams are difficult to understand and 20 percent said class diagrams are not understandable for customers.

Our goal is to model the requirements of the customer with real world entities well known to stakeholders. Figure 9 gives an example of a possible graphic representation on a level of entities in contrast to a modeling in UML class diagram. On the left hand a relation between Airport and Employees with a hierarchical structure is given in UML. Every Airport has many Employees and an Employee can be a manager of other Employees.

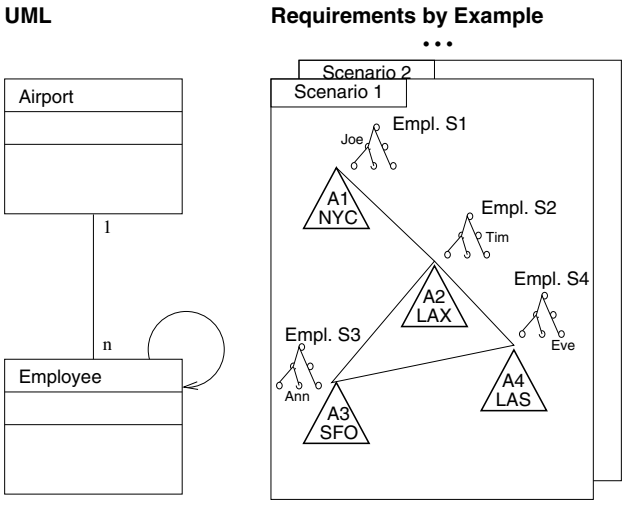


Figure 9. Requirements by Example

On the right hand a possible sample of *Requirements by Example* is given. So we have different Airports (A1 New York, A2 Los Angeles, A3 San Francisco, A4 Las Vegas). On every Airport Employees are working (Joe, Tim, Ann, Eve). It is possible to show multiplicities to customers. For instance, the number of Airports is visible for the customer. Also unfolded structures can be shown to the customers. For instance the customer can find the hierarchical structure or different roles of Employees. Furthermore, we can go beyond the limits of UML class diagrams. With the help of *Requirements by Examples* we can shown – for example by animation – which Airport is actually, on the instance level, connected with each other. Thus, distances, flight times between Airports, and so on can be easily specified. Consciously, we do not use UML object diagrams. As we shown before, object oriented representations are in general not understandable for stakeholder (from the point of view of over 60 percent of interviewees). Furthermore, UML does not afford an animation. So we need an understandable notation close-by stakeholders, like domain specific languages.

At the moment we work on the graphical appearance of symbols for *Requirements by Example*. Intuitive illustration facilities should be chosen. Last but not least we think about entities and structures to present scenarios (examples) with traceable values, to model beside multiplicities and unfolded structures, copies, recurrences, variants etc.

Regarding the general approach of UPEX (figure 8), we get first examples and information from the stakeholder in interviews or with the help of the paper prototypes. Then, by using an intuitive illustration we collect together with the stakeholder new entities and examples. During further iterations and refinements steps the static structure will be completed. Finally, this will give the developer a chance to raise the level of abstraction to a typed class structure.

Work Flow-Structures In addition to the static aspects of the system its dynamic aspects will also be described. In this case we use the CUTA [3] approach.

With CUTA (Collaborative Users Task Analysis) we can analyse processes on the stakeholder level in more detail in a practical and simple way. CUTA is a card game (see also [10]). Each card is divided in three categories. An activity, that describes a piece of work, its duration and frequency (see figure 10).

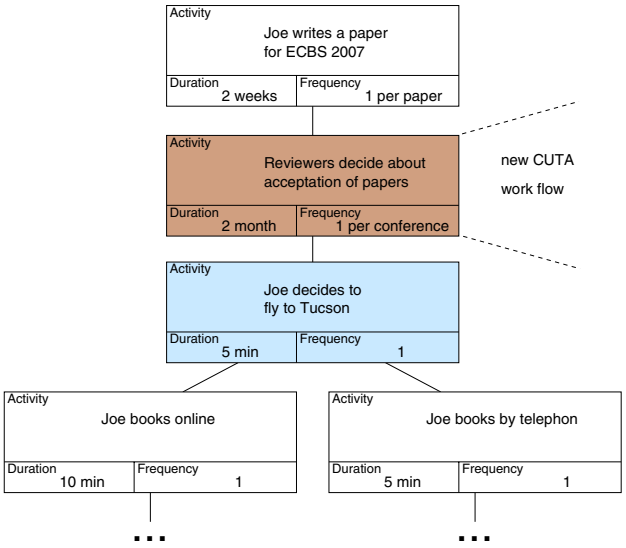


Figure 10. Typical CUTA sequence

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

The initial information from stakeholders is presented on cards with different significances and types of activities marked with tree different colors (for the original CUTA):

1. non-technology driven (Joe decides to fly to Tucson)
2. technology driven (Joe writes a paper)
3. non-object based (Reviewers decide about acceptation)

Together with the stakeholder the sequence of cards will be determined representing a sequence of activities in a specific work flow. During the card game the stakeholder may influence the model, e. g. by correcting activities or by asking for additional cards with new working steps, because of the hand-on nature of a game.

In such a way on the one hand we analyze the stakeholder's internal work flows or rather individual production steps that drive the system under development. On the other hand this will help the stakeholder to understand the dynamic behavior of the system, because of active discussions during the presentation of work flow alternatives. In this process a close participation of stakeholders takes place which raises the chance of detecting further work flows. So more "unusual" situations, for instance, error-handling problems can be found.

We keep in mind, that the description of dynamic aspects with CUTA is just example driven. The sequence of a domain work flow by example does not essentially reflect the underlying technological solution. To drive CUTA to a technological solution, we evaluated the card game by redesigning our existing online system for students to register for lectures. In every meeting we played a card game to determine one special feature of our system.

During these CUTA sessions we decided to apply changes to the original "type structure" of CUTA cards. We also established different kinds of arrows which differentiate between parallel activities and alternatives. In addition we assign an executing role to each activity and established notices.

In a next step we looked for an existing (semi-)formal notation which we can use as a target language to transform our types of cards into a developer centered notation. It turned out that work flows are a quite natural match. Because of the de-facto standard of UML we analyzed activity diagrams.

Currently, we work to define an automatism to transform our new CUTA model into UML activity diagrams. In a future step we will realize the automatism as a tool.

Visualization/Animation Layer Both parts, *Static Entity-Structures* and *Work Flow-Structures* (by example), will form the internal engine of the Animation Layer. With this Animation we want to visualize the stakeholder's processes by example, based on entities of the real world. So it should be possible to show, going back to our airport example (check figure 9), that the only way to fly from New York to Las Vegas goes via Los Angeles. This fact is as a consequence not only understandable for the development professional but also for a stakeholder on the end-user level. A validation of stakeholders requirements, especially of the modeled tacit knowledge, is possible.

At a lower level below the Animation a formal algorithm will be used to validate UPEX. Petri Nets as an abstract and formal basis for the developer could be a first choice for an internal background of the Animation.

5. Summary

The questionnaire gave us an insight in typical needs of enterprises to change the situation during the system development. This led us to the conclusion that the communication between developer and stakeholder needs improvement. The general concept of UPEX targets this need to gather and present the stakeholder's expertise. UPEX gives non computer scientist an understanding of the domain of computer science. Vice versa UPEX brings developer closer to the domain of his / her stakeholders.

At the present time we work on the *Static Entity-Structures* (by example). The concept of the *Work Flow-Structures* (by example) has been designed and rules for an automatism to UML activity diagrams are established. A tool for the transformation automatism is planned. Also a tool for the computer-aided paper prototype of the *Representation Layer* will be realized soon. Additionally to UPEX an online glossary was developed in which often used terms are explained for domain experts and laities to avoid misunderstandings. Last but not least the *Visualization/Animation Layer* has to be designed on both levels. Firstly, we will seek after different illustrations in the Animation Layer adapted for the stakeholder. Secondly, we will look to formal algorithms for a validation.

The development of UPEX takes place in two cycles. After a first prototypical implementation an evaluation together with farmers will take place and be used as direct feedback for further enhancement.

References

[1] I. Erfurth and W. R. Rossak. UPEX - User Participation by EXample. In *Proceedings of the Joint 10th European Software Engineering Conference (ESEC) and the 13th ACM SIGSOFT Symposium on the Foundations of Software Engineering (FSE-13), Doctoral Symposium*, pages 374–376, September 2005.

[2] K. Kirchner and I. Erfurth. Concept of a Farming Decision Support System for Cultivation Planning. In *Information*

Systems and People: Implementing Information Technology in the Workplace, 3rd itAIS Conference 2006, October 2006.

[3] D. Lafreniere. CUTA: A Simple, Practical, Low-Cost Approach to Task Analysis. In *Interactions III(5)*, pages 35–39, 1996.

[4] M. J. Muller. PICTIVE – An Exploration in Participatory Design. In *CHI '91: Proceedings of the SIGCHI conference on Human factors in com puting systems*, pages 225–231, New York, NY, USA, 1991. ACM Press.

[5] C. Sauer and C. Cuthbertson. The State of IT Project Management in the UK. Technical report, Templeton College, Oxford, November 2003.

[6] The Standish Group. `http://www.standishgroup.com`, 2006.

[7] The Standish Group International. The CHAOS Report. Technical report, The Standish Group, 1994.

[8] The Standish Group International. Extreme Chaos. Technical report, The Standish Group, 2001.

[9] The Standish Group International. 2004 Third Quarter Research Report. Technical report, The Standish Group, 2004.

[10] L. G. Tudor, M. Muller, T. Dayton, and R. Root. A participatory design technique for high-level task analysis, critique, and redesign: the CARD method. In *Human Factors and Ergonomics Society 37th Annual Meeting*, 1993.

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Ontological Traceability over the Unified Process[1]

Rodrigo Perozzo Noll
Pontifical Catholic University of Rio Grande do Sul – PUCRS
rnoll@inf.pucrs.br

Marcelo Blois Ribeiro
Pontifical Catholic University of Rio Grande do Sul – PUCRS
blois@inf.pucrs.br

Abstract

Traceability refers to the ability to link information in a process chain. This paper proposes the integration of ontologies into the Unified Process (UP) [1] to provide concept-based traceability throughout the software lifecycle. This approach allows the integration of the different models of a software system including business, requirements, analysis and design models in a lower granularity degree than conventional requirements traceability approaches. To assist the designers in creating the ontology and linking concepts to artifacts we provide a tool integrated with an UML modeler.

1. Introduction

The software development process comprehends the creation of different models. Models are abstractions of reality used to represent and control the architecture, evaluate opportunities and identify risks. Each model can be expressed over different views and abstraction levels and thus must be complementary. Each model is related to real world concepts. Using ontologies, we can specify the conceptual model that represents software information systems. This paper presents the use of ontologies to support the software development process, particularly the Unified Process. The aim is to integrate the different views that model the system concerning the business, requirements and analysis and design. Using ontologies, we can link these views and orthogonally map it across the concepts that represent the domain. We will present *ONTrace*, a functionality developed over ArgoUML [2] that supports this proposal.

The paper is structured as follows: Section 2 defines ontology and its relation with the Software Engineering and the Unified Process. In Section 3, we present the ontology integration into the Unified Process, describing the activities associated. Section 4 presents a traceability proposal and the developed tool to assist the process. Section 5 describes related works. Conclusion and future works, finally, are presented in Section 6.

2. Ontology and the Unified Process

The term "ontology" is borrowed from philosophy and means "a systematic account of existence" [3]. Ontologies are used to define the terms that describe and represent a knowledge domain. Ontologies include the definition of concepts and relationships among them, allowing knowledge to be shared and reused. Gruber [4] defines ontologies as "an explicit specification of a conceptualization". A conceptualization is an abstraction of some reality using a set of concepts, restrictions and relationships used in some application area.

Ontologies have been used to describe artifacts with different structures as taxonomies, metadata schemas or logic theories [5]. The Web Ontology Language (OWL) [5] is an ontology representation language which is partially mapped through description logic. Using OWL it is possible to process the information content and derive new information through the usage of inference. Ontology development also requires a systematic development approach as software development does.

2.1. Ontological Engineering

The literature provides some approaches for systematic ontology development. It was necessary to analyze those proposals [6] [7] [8] [9] [10] in order to

[1] Study developed by the Research Group in Intelligent Systems Engineering Group of the PDTI, financed by Dell Computers of Brazil Ltd. with resources of Law 8.248/91.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

extract the main aspects used to assist ontological engineering in the Unified Process.

Most of the proposals analyzed were developed using empirical methods. A simple comparison shows that all of them are based on an iterative process and somehow include the following steps:

1. Definition of the ontology scope;
2. Acquisition of domain concepts (conceptual model);
3. Formalization of the concepts (formal model);
4. Integration to existing ontologies;
5. Definition of the axioms;
6. Validation.

Some approaches emphasize some steps most than others, but all of them indicate that these six main steps are needed.

According to [8], there is no single correct way to model a conceptual domain, there are always viable alternatives. Most of the effort related to build an ontology is related to the domain comprehension and the conceptualization. During the earlier phases of the Unified Process, most of the efforts consist on the same matter: business modeling and domain definition. Traditional software development processes do not use logic languages to represent a conceptualization formally.

It is possible to use some artifacts of the Unified Process to reduce the effort related to concepts identification as described on [6] as the Software Requirements Specification or the Use Case Descriptions. A new UP discipline can link the concepts modeling through an ontology definition with the other artifacts produced by the other disciplines.

2.2 Ontology and Software Engineering

Although ontology engineering and software engineering concentrates on concepts mapping and translation among models to construct a relevant computational representation, there are some important differences between them. The complexity related to a logic model is greater than the related to an oriented-object (OO) model. For instance, the predicate logic definition implies that the relationships among concepts are considered a first-order element, and in OO they are subordinated to the class concept. It is not possible to map a full ontological model, with all logic restrictions, into an OO model because OO representation only captures the concepts taxonomy and relationships.

2.3 Ontology and the Software Development Process

A software development process defines a software product lifecycle. The lifecycle comprehends all the activities related to the software product production, since the requirements analysis until its development, testing and maintenance.

The UP is an example of a well-accepted software development process. UP organizes the software development tasks into four distinct phases: inception, elaboration, construction and transition. Each phase could support many iterations, each of them is categorized into nine disciplines: business modeling, requirements, analysis and design, implementation, test, deployment, configuration and change management, project management and environment.

During the Business Modeling discipline, the UP suggests the development of a conceptual representation of the real world concepts (the problem domain) called the Domain Model. The Domain Model describes the concepts of a domain and the relationships among them using the UML class diagram syntax.

It is important to observe the similarity between ontologies and the Domain Model: both specify concepts and their relationships in the context of a domain of discourse. The Domain Model represents an application domain using a business view instead of an architecture view, with the aim to understand the system context and share this conceptualization between the stakeholders. Ontologies can also be used to represent a shared conceptualization among the stakeholders of a software development project while representing the application domain in a business perspective.

In the Requirements discipline, the team understands the user's needs, eliciting with the domain specialist the software characteristics and restrictions. To develop a consistent ontology for a domain, it is necessary to obtain explicit knowledge about the domain. Using the domain model as a source it is possible to create an ontology for the domain and refine it using the software requirements specification. This refinement should identify new concepts, organize the taxonomy and define the restriction rules that represent the problem domain.

After the requirements specification, the software will be modeling and its architecture will be defined in the Analysis and Design discipline. The use of ontologies on the earlier phases of the development process allows the semantic traceability between the concepts in the domain and the architecture elements that are derived from them.

To foster semantic traceability, this paper presents a proposal for ontology integration into the Unified Process. The Unified Process was chosen because it is well-accepted both in scientific and industry communities. Using ontologies, we can generate traceability links between the business model, requirement specifications and analysis and design models. More over, using inference over ontologies we obtain new knowledge that represents implicit and cognitive links among the system's artifacts.

3. Ontology Integration into the Unified Process

Our approach to integrate ontologies in the UP is non intrusive since it does not modify the existing UP disciplines. The basic idea is to create a new discipline called *Ontological Engineering*. This discipline produces a logic model to represent the software domain. Using this model, we link all the software specification elements with the concepts in the ontology.

3.1 Ontological Engineering Discipline

This proposal suggests the integration of an ontological model to the traditional software development process. The Ontological Engineering discipline captures good practices identified by many ontological engineering approaches, however focusing on software development. All the analyzed proposals are only focused on the engineering of a logic model for a certain domain not directly related to software engineering.

To represent the discipline's workflow, the SPEM (Software Process Engineering Metamodel) [11] notation will be used. Table 1 presents the elements used and their meaning.

Table 1. SPEM elements used to define the discipline's workflow.

Element	SPEM Icon	Element	SPEM Icon
Activity		Process role	
Phase		Work definitions	
Process		Work product	
Process package			

We will use some UP artifacts as input to ontological engineering and its activities are based on the steps presented in Section 2.1. The discipline is structured over three major activities: Design, Maintenance and Validation of ontology. The ontology is designed in the earlier phases of the discipline (executing steps 1, 2 and 3). The design continues until the Domain Model is finished. Once it is finished, the ontology can be extracted directly and maintenance activity can start (executing steps 4 and 5). Every modification in the ontology must be validated (step 6). The workflow diagram in Figure 1 shows the discipline's structure.

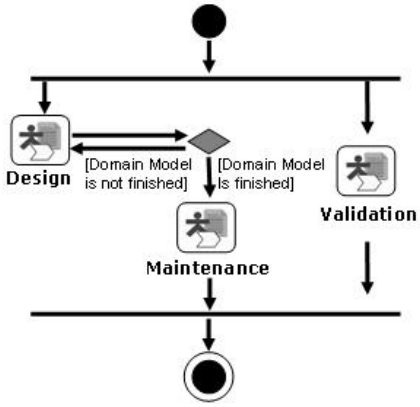


Figure 1. Ontology Engineering discipline workflow.

To give an idea of the effort involved in the new discipline execution, Figure 2 shows how the ontological engineering is dispersed over the UP phases. It is possible to notice that ontological engineering activities start after the Business Modeling discipline, when the designers have a good understanding of the problem domain and can represent the concepts involved with the software under development. Throughout the software development, the ontology must be updated to maintain its integrity to the new knowledge discovered when detailing the software architecture.

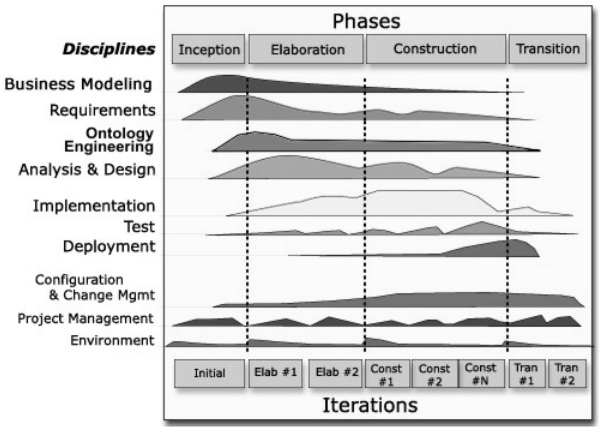


Figure 2. Unified Process extended.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

The following sections will detail each activity presented in the discipline's workflow.

3.1.1 Design. The design activity defines the ontology scope and its primary concepts. This activity also provides support to taxonomic and non-taxonomic properties identification.

Figure 3 presents the process model related to the design activity. The *Ontology Engineer* role indicates the responsible to perform the activities related to the ontological engineering discipline. The Ontology Engineer takes as input the Domain Model and the Requirements Specification and updates the Domain Model focusing on the ontology development. The modifications include the concepts refinement and the taxonomic and non-taxonomic properties (*Object-Properties* and *Datatype-Properties*) definition. After the Domain Model is matured and finalized, the next activity will extract a preliminary version of the ontology from this model, generating a new work-product called Ontology.

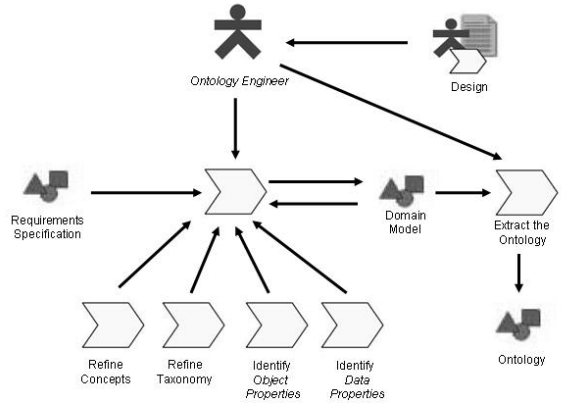


Figure 3. Design Activity.

The preliminary ontology version is built using the mapping guidelines presented in Table 2. This guideline preserves the semantic between the two representations. We will adopt OWL language to represent the ontology since it is officially recommended by W3C [5].

Table 2. Mapping between UML and OWL.

UML	OWL
Class	OWL Class
Attributes	Datatype Property • *Domain*: Parent class • *Range*: XSD defined by XML Schema.
Associations	Object Property • *Domain:* Source association class. • *Range:* Target association class. • *Restrictions:* 1. If the initial association and the final association are navigable, so the property is symmetric. 2. If the multiplicity of both associations is the same, generate a new cardinality restriction. If not, create a maximum and minimum cardinality restriction.
Generalization	Defines the taxonomy between classes.

The complete mapping between an ontological model and an oriented-object model is not possible due to the lack of support in OO models to represent restriction rules. However, the inverse is not true. We can map an oriented-object model to a logic model and further add restriction rules to the domain.

3.1.2 Maintenance. The design activity does not produce a complete version of the ontology, because it does not present most of the restriction rules. The ontology maintenance could be done using the approach provided by [9], i.e., to identify on the system specification all the concepts related to the domain and refine the ontology manually. The first step is to observe the taxonomic organization (generalization/ specification) and refine it. Then, the non-taxonomic relations are inspected and refined (association and attributes).

The main activity's goal is to generate traceability links between model elements (requirement specification, analysis model and design model) and the concepts of the ontology. During the development process, all the artifacts and its elements could be orthogonally mapped through the ontology concepts. Additionally, these links will represent the knowledge related to the software development process. Later we will present how these links are found.

Figure 4 presents the Maintenance activity. Like the Design activity, the Ontology Engineer is the role responsible for its tasks. The activities within the maintenance main activity will receive as input: the Ontology, the Requirements Specification, the Analysis Model and the Design Model. The Ontology Engineer will update the ontology throughout the whole development cycle, adding traceability links to the model elements and refining the concepts, properties and restrictions rules.

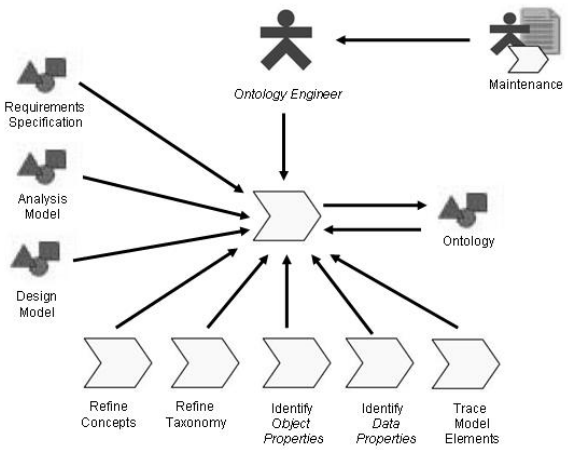


Figure 4. Maintenance Activity.

3.1.3 Validation. The Validation activity provides the constant evaluation of the ontology integrity. It is based on the approaches defined in [9] and [10]. For each development lifecycle iteration validate if the logic model still consistent using an inductive and pragmatic approach. The validation helps in generating a consistent and coherent representation of the application domain. This process also proposes three validation types for each part of the ontology development: unit test, integration test and acceptance test. The two first should be executed by the Ontology Engineer and the last should be executed by software agents or other applications that uses ontologies.

4. Semantic Traceability

On the previous section a process to develop an ontology that represents the conceptual model of an information system was presented. In the process we explicit define all the concepts and the relationships among them, including taxonomic and non-taxonomic properties. However, the main aim to develop an ontology is the possibility to relate all the artifacts and their elements to the concepts during the software development process.

Using ontologies, we can recover implicit traceability links using an inference engine. To allow the concept-to-artifact mapping, a general structure exemplified in Figure 5 is proposed. The ONTrace concept is a class that represents an UML element instance (such as use cases, classes in a class diagram, etc) generated in the software development. This resource contains a datatype property called *ontraceRecover* that relates the element to the ontology concepts (represented as OWL classes).

This traceability structure is included on the ontology generated from the domain model and, for each traceability link, we create or update a new instance of the resource ONTrace. These instances are linked to the ontology concepts that it will be used as the index for recovering the traceability links. Figure 5 presents the structure instantiated for an example using the notation provided by the OMG [12] to represent ontologies.

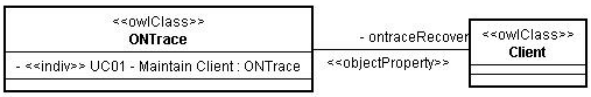


Figure 5. Ontological Structure for Traceability.

The example is extracted from an e-commerce scenario. Suppose that the use case "maintain client" is related to the ontology concept "client". Another use case "get product information" is related to the ontology concepts "client", "employee" and "product". So, it is possible to say that "maintain client" and "get product information" are related through the "client" concept (a direct relation link).

You can relate the entire system (models and their elements) to the ontology concepts and recover all of them later through direct and indirect relations. In the same example, suppose that there is a use case called "maintain employees" and it is related to the ontology concept "employee". With this configuration it is possible to infer new information about the artifacts: the use case "maintain client" is related to the use case "maintain employee" because the ontology concepts that relate both are associated through a property (considering that *getInformation* property relates *employee* and *client* concepts). This property indicates that a conceptual item is related to another and thus the elements linked to them could impact each other. Summarizing, we could say that since:

- The use case "maintain client" is related to the ontology concept "client";
- The use case "maintain employee" is related to the ontology concept "employee";
- The ontology concept "employee" is related to the ontology concept "client";

It is possible to say that the use case "maintain client" is related to the use case "maintain employee". This relationship is indirect and could be recovered by restriction rules, justifying the use of ontologies for traceability.

Figure 6 presents the traceability idea: the possibility to create ontology's individuals that relate the business, analysis and design system views to ontology's concepts.

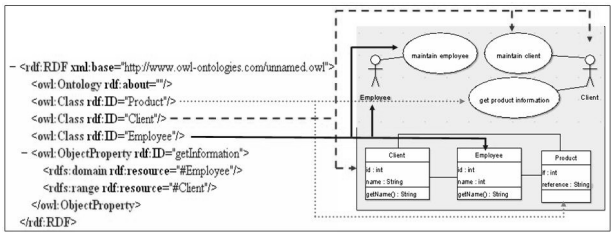


Figure 6. Semantic Traceability Overview.

To support the ontological engineering process and the semantic traceability, a tool was developed over ArgoUML, called *ONTrace*.

ONTrace automates the ontology generation from the Domain Model using Table 2 criteria. The ontology model could be exported and imported from ArgoUML+ONTrace at any time. It is possible to edit the ontology, adding all the restriction rules needed to consistently represent the domain using some ontology editing tool, like Protègè [13]. After editing, you can import it to the tool and continue to add traceability links.

Based on the ontology generated, a panel at the bottom of the tool is populated with all resources of the ontology. The resources include OWL classes, datatype properties and object properties. For each element of the model you can link all the resources it is related to, only checking the box beside its name. Every time you check a box, you create or update an individual of the ontology that represents a traceability link.

To recover the traceability links the user can query the ontology using an inference engine. This feature recovers elements related to the same concept and to related concepts, presenting them on the "Related Elements" in the interface. The relationship recovery degree, that is, the number of concept-to-concept relationships inspected for artifacts recovery is configurable. In this study we worked with recovery degree two: concept that relates to another concept. Figure 7 presents the query interface for *ONTrace*.

5. Related Work

The related work is divided into the two basic aspects of this paper: ontological engineering and semantic traceability.

Ontology engineering presents a lot issues raised when structuring knowledge. We have already presented ontology engineering methodologies in the Section 2.1. Basically, these work influenced our proposal in terms of the good practices they recommend. In essence the main focus in this paper is the generation of an ontology to support the software development process. Other works provide guidelines to support ontology generation for any domain, but lack specific annotation to support artifacts traceability.

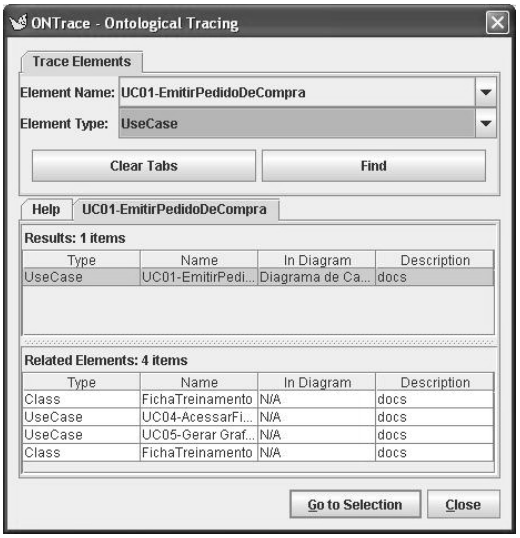


Figure 7. Query Interface for ONTrace.

In terms of traceability, most of the works propose a traceability matrix relating high-level or detailed requirements of the software product to the artifacts generated in analysis, design and test workflows. A traceability matrix is a table that correlates any two baselined documents. We can found some proposals based on requirements traceability using a matrix and variants at [14], [15] and [16].

Correlating the concepts of the problem domain to the software architecture provides an orthogonal traceability if we compare to the traceability matrix of other approaches. The software development is a process of abstraction decomposition from a total conceptual representation to a formal, computable one. The concepts involved in this process are always presented in the different representations of the system. Using concepts to trace the artifacts is a straightforward thinking for this scenario. By doing this, the traceability granularity is reduced allowing a better matching between related artifacts than using the whole requirement as the link basis.

6. Conclusion and Future Work

Traceability is an important issue in software development. Many works have been done to enhance traceability between developed artifacts throughout the development lifecycle. This paper introduced a new UP discipline called Ontological Engineering. The idea is to integrate an ontology that maps the domain concepts and indexes all the artifacts produced. This allows the semantic traceability through inference over the ontology. As the result of the inference execution,

the artifacts related to the concepts indicated as base facts to run the inference engine are returned. This approach can reduce the traceability granularity, providing richer links than those provided by methodologies based only in the requirements as the base for indexing.

We are currently working in analyzing our modified process in terms of usability using proposal Experimental Software Engineering [17] techniques. Another paper will compare traceability using our approach against traceability using a traceability matrix.

10. References

[1] Jacobson, I., I., Booch G. and Rumbaugh, J. "The Unified Software Development Process". Addison-Wesley Professional, 1999.

[2] *ArgoUML - A UML design tool*. Available at: <http://argouml.tigris.org/>.

[3] Gómez-Pérez, A. and Benjamins, V. R. "Overview of knowledge sharing and reuse components: Ontologies and problem-solving methods". In International Joint Conference on Artificial Intelligence (IJCAI-99), Workshop on Ontologies and Problem-Solving Methods (KRR5), V.R. Benjamins, et al., Editors. 1999: Stockolm, Sweden.

[4] Gruber, T.R. "Towards Principles for the Design of Ontologies Used for Knowledge Sharing". International Journal of Human and Computer Studies, 43/(5/6):907-928. 1993.

[5] *World Wide Web Consortium*. Available at: <http://www.w3.org>.

[6] Gruninger, M. and Fox, M. S. "Methodology for the Design and Evaluation of Ontologies". Workshop on Basic Ontological Issues in Knowledge Sharing, IJCAI-95, Montreal.

[7] Fernández, M., Gómez-Pérez, A. and Juristo, N. "Methontology: From Ontological Art towards Ontological Engineering". In Proceedings of AAAI97 Spring Symposium Series, Workshop on Ontological Engineering, pages 33--40, 1997.

[8] Noy, N. F. and McGuinness, D. L. "Ontology Development: A Guide to Creating Your First Ontology". Technical Report KSL-01-05 and SMI-2001-0880, Stanford Knowledge Systems Laboratory and Stanford Medical Informatics, March 2001.

[9] Sure, Y. and Studer, R. "On-To-Knowledge Methodology – Final Version". On-To-Knowledge EU IST-1999-10132 Project Deliverable D18 (WP5). Institute AIFB, University of Karlsruhe, 2002.

[10] Kishore, R., Zhang, H. and Ramesh, R. A. "Helix-Spindle Model for Ontological Engineering." Communications of the ACM, February, 2004/Vol. 47, No. 2

[11] *Object Management Group*. Available at: <http://www.omg.org>.

[12] *Ontology Working Group*. Available at: <http://ontology.omg.org>.

[13] *The Protègè Ontology Editor and Knowledge Acquisition System*. Available at: <http://protege.stanford.edu/>.

[14] Jarke, M. "Requirements Tracing". Communication of ACM, vol.41 (12), 1998. pp.32-36.

[15] Ramesh, B. and Jarke, M. "Towards Reference Models for Requirements Traceability". IEEE Trans. Software Engineering, vol. 27(1), 2001.pp.58-93.

[16] Egyed, A. and Grünbacher, P. "Automatic Requirements Traceability: Beyond the Record and Replay Paradigm". In 17th IEEE International Conference on Automated Software Engineering (ASE), Edinburgh, Scotland, UK, September, 2002. Proceedings.pp.163-171.

[17] Wohlin, C., Runeson, P., Höst, M., Ohlsson, M., Regnell, B. and Wesslel, A. "Experimentation in software engineering: an introduction". Kluwer Academic Publishers, USA, 2000.

A Service-Oriented Extension of the V-Modell XT*

Michael Meisinger
Institut für Informatik
Technische Universität München
Boltzmannstr. 3, 85748 Garching, Germany
meisinge@in.tum.de

Ingolf H. Krüger
Department of Computer Science
University of California, San Diego
La Jolla, CA 92093-0404, USA
ikrueger@cs.ucsd.edu

Abstract

The ever growing size and complexity of both technical and business systems requires efficient software engineering approaches to keep development cost under control while still being able to finish development efforts in time with the required functionality and quality. Systematic software and systems engineering approaches help to push the boundary further and leverage the complexity on many different levels. On the one hand, the availability of appropriate models and notations for the systems under development throughout the development cycle and for all levels of abstraction helps to understand and modify manageable views of the system. On the other hand, systematic development processes can provide the harness for successful project execution and for the ability to repeatedly create results that meet the required quality standards and functionality within the budgeted cost and time. In this paper we combine a proven generic project management framework with a methodology for developing complex multi-functional systems. We embed our service-oriented development *approach for reactive systems into the system development process model* V-Modell XT *by providing a modular extension of the V-Modell XT for service-oriented development. We introduce our development approach by means of a running example from the complex control systems domain, the BART traffic controller example.*

1. Introduction

Enabled by wired and wireless communication technologies, traditional business intelligence and technical systems increasingly converge into ultra-large scale (ULS) systems. Examples of ULS systems include avionics, automotive, command and control, as well as telematics and public safety systems, to name just a few. In all these domains, the primary challenge to software and systems engineering is the integration of a wide variety of subsystems, their associated applications, data models and sources, as well as the corresponding processes, into a high quality system of systems under tight time-to-market, budget, security, policy, governance and other cross-cutting constraints.

Modern ULS systems also have a number of other challenging requirements characteristics that all but exclude monolithic software and systems architectures: ever changing business processes, demands at including both legacy and emerging systems as they become available, and the need to cater to changing requirements, are some examples.

These requirements characteristics have led to a high demand for loosely-coupled integration architectures [16]. As a mechanism to achieve coupling, the notion of *service* has attracted increasing attention both in industry and academia. Web services [32, 39] have emerged in the business domain as an attempt at simplifying distribution, publishing, discovery, addressing, and accessing of software functions across the Internet. The increasing complexity of embedded software has led to a similar trend in automotive and similar technical system domains [21, 1].

Intuitively, web services are application programming interfaces (APIs) that project a subsystem's capabilities to the infrastructure via a well-defined Internet address. It is this *deployment* character of service-oriented development that is behind the recent popularity of the service concept; popular open standards such as HTML and SOAP, together with their integration in Integrated Development Environments (IDEs) have made it virtually effortless to expose subsystem functionality in terms of a web service.

However, the essence of a comprehensive service-oriented software and systems engineering approach cannot be deployment-centric alone. Rather, the idea of structuring an ULS system into a collection of services that project ca-

*V-Modell is a registered trademark of the Federal Republic of Germany. Our work was partially funded by the Deutsche Forschungsgemeinschaft (DFG) through the project *InServe*. Further support was provided by the UC Discovery Grant and the Industry-University Cooperative Research Program, as well as by funds from the California Institute for Telecommunications and Information Technology (Calit2). We are very grateful to Massimiliano Menarini for joint work on this subject.

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

pabilities of (sub)systems carries much farther than what is addressed by the question of how a service – once identified – can be implemented on the infrastructure.

This paper, therefore, attempts a seamless integration of the concept of services into all phases of the ULS systems development cycle: from logical architecture design to implementation/deployment and maintenance. In general, complex system development requires a systematic approach. Development methods and processes, as well as process and capability models have proven to support the development efforts in the aim for better adherence to time and cost budgets, functionality delivered, quality of the results and repeatability of the entire process. All these processes, models and methods have different properties and application areas, for instance, providing detailed software development methodology support, establishing essential project management practices and helping to comply with specific regulations and standards. They range from quite heavy-weight, sophisticated processes to very light-weight, dynamic agile ones. The particular choice depends on many factors and requires competent and wise project management decisions. In general, each process and method should only be applied with prior adaption to the specific project and organization, and checked for effectiveness during the course of a project.

Problem Definition and Solution Overview Because services are projections of capabilities of (sub)systems, they are fundamentally *partial* in nature. It is the composition of a set of services that yields the desired integration architecture. Most established development processes, however, focus on the notion of component as the unit of development and deployment – this results in a poor match with the partial nature of the service concept.

A central goal of the approach we promote in this paper is to cleanly separate a logical notion of service, which captures the capability that emerges from the interplay of a set of components, from its mapping to a concrete deployment architecture. In short, services are the centerpiece of logical architecture design, whereas components implement the services at runtime. We employ mechanisms of *model-based development* to accomplish this separation. Logical models describe capabilities in terms of functionality, distribution of components and quality properties independently of any implementation details or deployment architecture design decisions. The implementation models contain these details and decisions, and must be a consistent refinement of the logical models. This distinction has the advantage that many implementation models exist, which satisfy one logical model and that implementation models can be developed strictly after the logical models.

Technical complexities, such as defining the functional behavior of the system by identifying the services and their dependencies, designing objects/components and their interfaces so that they can provide the services with the required quality properties, and deploying them on a given middleware for most efficient operation are only part of the challenge. Organizational requirements within the project, the need for consistent documentation and efficient tool support, and extended system maintenance time spans impose further challenges. Development processes provide systematic support in some or all of these regards to various degrees.

The V-Modell XT [4, 30], for instance, is a modern software and systems development standard covering project management, engineering and supporting processes. It provides a generic process model, which is easy to understand and to use, and flexibly adaptable to the needs of organizations and projects. The V-Modell XT promises to lead reproducibly to project results of higher quality with less cost and resources spent. It is generic in nature and does not provide detailed methodic instructions for systems development; it has a modular setup enabling flexible extension.

Our goal is to provide an adapted, widely accepted development process that is flexible enough to support our service-oriented development models through all stages of system development and maintenance, without restriction to certain application domains. This will increase the applicability of our approach and help to gain wider acceptance beyond the research community. In this paper, we will present an integration of our service-oriented development approach with the V-Modell XT, which fulfills the requirements we have presented above and can be tailored to a wide range of development processes for ULS systems.

Service-Oriented Specifications The development methodology we propose in this paper focuses on services as first class entities throughout the development process of ULS systems; it establishes a clean separation between the services provided by the system under consideration, and the architecture – comprised of components and their relationships – implementing the services. Our approach is well suited for process embedding and tool support.

Following [14, 20] we use the notion of *service* to decouple abstract behavior from implementation architectures supporting it. Typically, services coordinate workflows among domain objects; they may also call, and thus depend on, other services. In this sense, services are specializations of use cases to specify interaction scenarios; services "orchestrate" the interaction among certain entities of the system under consideration to achieve a certain goal [6]. In contrast to use cases, which describe functionality typically in prose and on a coarse level of detail, a service is defined via the interaction pattern among a set of collaborators required to deliver the functionality. Services are partial interaction specifications.

As methodological core ignoring any management and QA aspects for now, we employ a two-phase, iterative development process (see [14]). In the first phase, services are elicited from use cases, and captured in terms of interaction patterns. In the second phase, a deployment architecture is defined as a set of interacting components; then the services are mapped to the deployment architecture to yield the overall software and systems architecture.

The relevant use cases and their relationships are defined as use case graph. This yields a relatively large-scale, scenario-based view on the system. From the use cases, sets of *roles* (actors) and *services* (functions) as interaction patterns of roles are derived. Using roles decouples from interaction details, because roles abstract from components or objects. Roles describe the contribution of an entity to a particular service independently of what concrete implementation component will deliver this contribution. An object or component of the implementation typically will play multiple roles at the same time. The communication relationships between roles are captured in a *role domain model*. The set of services is *mapped* to a *component configuration* refining the *role domain model* to yield an *architectural configuration*. These architectural configurations can be readily implemented, for instance prototypically by code generation.

Contributions and Outline As main contribution, this work presents an integration of a systematic approach for the development of ULS systems into an established and widely accepted generic process standard. We explain our modeling methodology and present its application by means of a running example through all stages of development. We show how we integrate this approach into the existing large-scale development model, V-Modell XT.

In Sect. 2, we introduce a service-oriented model of our running example, the Bay Area Rapid Transit system (BART). In Sect. 3, we explain the basic concepts and extension mechanisms of the V-Modell XT, which we integrate with our service-oriented modeling approach in Sect. 4. In Sect. 5, we discuss advantages and shortcomings of this integration. Sect. 6 contains related work and Sect. 7 presents conclusions and an outlook.

2. Service-Oriented Development of Complex Systems

The systems we address with our service-oriented approach are complex distributed systems. The complexity we refer to here stems from the need to integrate multiple different parts whose interplay is difficult to grasp with traditional techniques. Rather than treating component interplay as an afterthought, addressed only during late stages of deployment and integration, we focus on *services*, defined as the interaction patterns among roles, throughout the development process.

The BART Case Study The BART case study [41] describes parts of the Advanced Automatic Train Control (AATC) of the Bay Area Rapid Transit (BART) system. BART is the San Francisco area, heavy commuter rail train system. The case study describes the part of the train system that controls speed and acceleration of the trains. BART was previously used as a case study in the area of distributed systems and for the application of formal methods [11].

The BART system automatically controls over 50 trains on a large track network. Manual operation of the train control is limited mostly to safety issues and to cases of emergency or malfunction. The AATC system controls the train movement with a requirement to optimize train throughput, while constantly ensuring safety. The AATC system operates computers at the train stations which each control a local part of the track network. Stations communicate with the trains via a radio network and with neighboring stations. Trains receive acceleration and brake commands from the station computers and feed back speed, position and status information.

We focus on modeling the reactive behavior of a station computer and of the trains. We apply our service-oriented development approach to identify the different services of the system and to specify a service model that will help us to design an effective service-oriented architecture. Our model-based approach results in a high level design model of the AATC that can be refined systematically into an implementable system. In the following, we will explain the steps in more detail (see also [15]).

Use Case Elicitation From the requirements [41], we identify a number of use cases, for instance, "A *train* communicates its current status to the responsible *control station*". Each use case can be broken down into more detailed steps, leading to a comprehensive use case view of the BART system. Analyzing the use cases leads to an initial list of actors, or *roles*; in our case these are Train, Control Station, the Safety Computer (VSC) and an External Data Source as actors. We depict roles and their communication relations in an initial role domain model, as shown in Fig. 1.

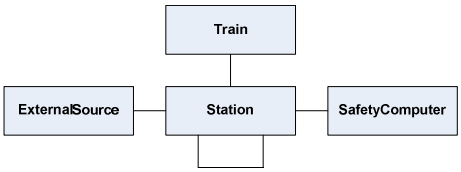


Figure 1. High Level BART Roles

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Modeling Services and Roles We model roles and services, starting with the above initial role domain model, by systematically going though the list of use cases and identifying interaction patterns that define services. The services we identify are somehow a refinement and formalization of the use cases. In the process of identifying interaction patterns, we may identify further actors; we add these as roles to the role domain model. Finally, after modeling all services the resulting role domain model looks as depicted in Fig. 2.

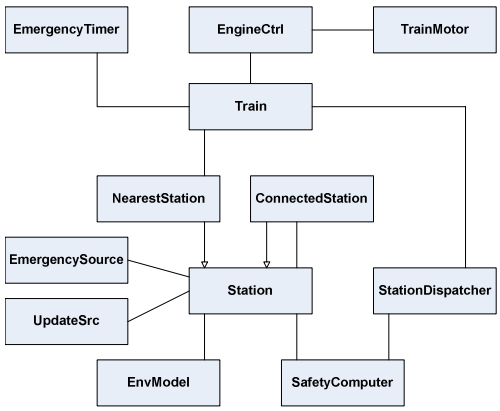


Figure 2. BART Role Domain Model

We use the extended MSC notation of [13, 15] to specify services. This notation is based on the Message Sequence Chart [9] standard and provides an intuitive graphical language for specifying interaction patterns and is well-accepted among engineers. Extensions to the standard notation were cautiously made based on a formal semantics to provide increased expressiveness and more powerful operators suitable for modeling service-oriented systems. To model the services, we make use of our tool-chain introduced in [1, 5].

We capture the interactions between the station computer (and its subcomponents) with a train (and its subcomponents). Other entities, such as external data sources, are part of the interactions as well. When modeling the interactions, we abstract from any concrete deployment architectures and do not consider multiple occurrence of the same entities. For instance, we specify the interactions between a train and the station computer abstractly, not knowing how often this interaction will occur in the implementation.

Good design principles suggest a hierarchical design of the service model. We use High Level MSC (HMSC) to introduce hierarchy. Intuitively, an HMSC is a graph depicting a *flow* through a set of services. The HMSC "TrainLoop" in Fig. 3 shows an infinite flow of activities of regular train operation, potentially preempted by exceptional behavior in case of an emergency situation; then the emergency is resolved after which the behavior returns to normal operations.

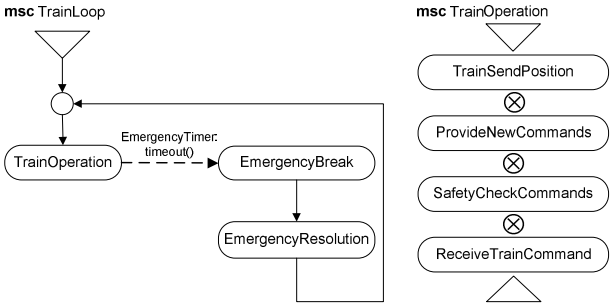


Figure 3. HMSCs for BART services

The "TrainLoop" HMSC contains MSC references, depicted by labeled rounded boxes, pointing to more detailed interaction behavior. The referenced functionality for "TrainOperation", for instance, is specified in the MSC shown on the right side of the figure. It shows a composition of four services by means of the "join"-Operator, depicted as $\otimes$. The semantics attached to the join of two services is the interleaving of the two behavior specifications, synchronized on common messages. The *join* operator is a powerful means to combine and synchronize *overlapping* services – this ability to disentangle service specifications is central in our approach. We call services overlapping if they share at least two roles and at least one message between shared roles. Overlapping interactions will occur only once – synchronized – in the resulting behavior. For details about the join operator, see [13, 19, 14]. We can also apply operators for *Sequence* of interactions, *Non-deterministic choice*, *Parallel* interactions and for *Preemption*, defining exceptional behavior.

Fig. 4 shows the behavior specification of a train sending current status values to the nearest station that processes the information in the syntax of an extended Basic MSC. Messages are depicted as horizontal arrows between two roles (represented as vertical axes labeled with the name of the role) and have parameters to indicate transmission of data values.

We make use of MSC operators, depicted as labeled boxes, to express repetition and choice in the interaction flow. The *LOOP*$<*>$ box around all the interactions in the MSC expresses repetitive behavior. Alternative or optional behavior is expressed by *ALT* boxes. Different alternatives are separated by horizontal dashed lines through the box. To indicate optional behavior, we leave one of the alternatives empty. Further operators at our disposal are for *parallel*, *joined* and *preempted* interactions. With *TRIGGER*, we express liveness conditions. State markers (hexagons) define states in the execution of a role. For more information and precise semantics definitions, see [13].

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

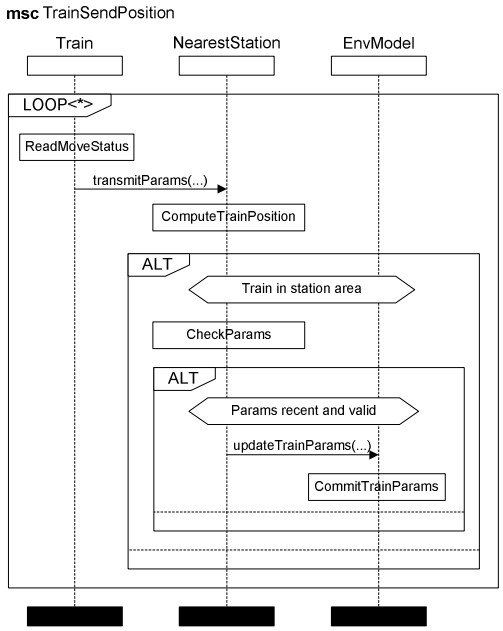


Figure 4. MSC TrainSendPosition

We integrate control aspects into reactive interaction specifications by means of *local actions*. Local actions are depicted as labeled boxes on role axes. The meaning of this syntax is that a role performs an activity based on the information available until this point in time. Information can be local variables, data previously received via messages and the role state.

Mapping the Service Model to Components When transitioning from a service model with roles and interactions to an implementable architecture, we define *component types*, which are blueprints of *component instances* in the architecture. We define their communication interfaces to other component types and the services they implement. The component model needs to be a refinement of the structural role dependencies. Table 1 shows an example role-to-component mapping. The behavior of the component types can be derived for instance from the service specifications using the component synthesis algorithm, described in [20].

Component Type	Role	Description
FastCPU	Station EnvModel StationDispatcher	A fast CPU computer for operative station control
SlowCPU	SafetyComputer	High Reliability (MTBF) slow CPU unit checking safety conditions
Train	Train TrainMotor EmergenyTimer EngineCtrl	Train computing unit on board of a train
Interlocking-System	UpdateSrc TrainMotor	Interlocking system, controlling switches and gates

Table 1. BART Role Mapping

Defining a Component Architecture Fig. 5 shows a simple architecture, which implements the service model that we have specified above. It shows the structure of the system's components and their connections. Components are instances of component types and can occur multiple times in a system configuration. Each instance has a defined name and specific type.

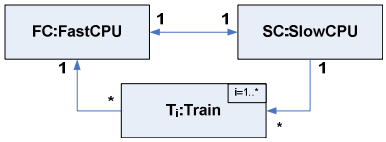


Figure 5. BART Component Architecture

3. The V-Modell XT

V-Modell XT Concepts The V-Modell XT [4, 30] is a modern German software and systems development standard; its application is mandatory for all contractors supplying IT systems to the German federal government and military. The V-Modell XT has a modular setup and enables flexible usage and extension. In order to extend the V-Modell XT it is imperative to know its main concepts, which we explain in the following:

- *Work Products* are the essential project results and artifacts (documents, models, code, deliverable systems). They have a prescribed structure and content and can be structured further into subjects (subsections). Work products have a responsible role and will be quality evaluated.
- *Product Dependencies* define consistency relations between the contents of different work products. This will keep new work products in a project consistent to existing ones, assuring overall product quality and information traceability.
- *Activities* define actions that need to be performed in order to edit work products. One activity is associated exactly with one work product. Activities can be structured further into sub-activities.
- *Roles* describe profiles of responsibility for individuals working in a project.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

- *Process Modules* group Work Products, Activities, Roles, and other V-Modell XT elements into self-contained units with a common purpose such as project management, requirements management, systems development, etc. Process modules may depend on others. They can be understood, applied and modified independently and are the units of tailoring and extension of the V-Modell XT.
- *Tailoring* is the process of adapting the V-Modell XT to a specific project or organization, by selecting the suitable process modules out of the repository of available ones. Tailoring results in a seamless consistent adapted software development process.

Systems Development The V-Modell XT defines many processes which are important for the development of systems, including management processes (such as "Offer Preparation and Contract Fulfillment"), engineering processes (such as "Specification of Requirements"), as well as supporting processes (for instance "Quality Assurance"). Each of these processes is defined in a process module. In the following, we concentrate on the for main system development process modules:

- *Specification of Requirements*: The process of preparing a requirements specification document based on a project proposal and evaluating these requirements in terms of effort, cost and importance.
- *System Development*: The process of decomposing a complex system into manageable units of software, hardware, supporting systems and additional materials (such as manuals) and of integrating the units to the deliverable system.
- *Software Development*: The process of developing an individual unit of software, which includes specification, software architecture design, implementation and integration, test specification and unit evaluation.
- *Hardware Development*: The process of developing an individual unit of hardware.

These four process modules define work products (artifacts), activities and roles that are required to develop a system from customer defined requirements specification to the acceptance of the final deliverable. Fig. 6 depicts the different stages of development in the shape of the "V", which is often associated with the V-Modell XT. System development involves the creation of a number of significant project results with clearly defined role responsibilities. Activities describe how the project results are created.

Requirements are collected in the *Requirements Specification* by the *Requirements Engineer (Acquirer)*. The document contains a situational overview, a list of functional and non-functional requirements, the list of deliverables with

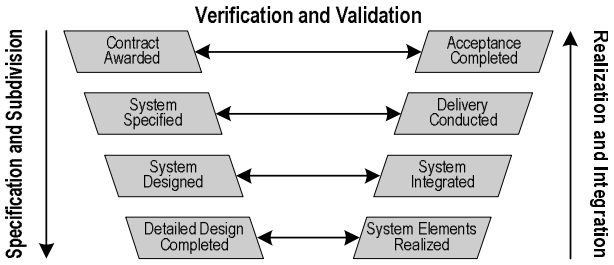


Figure 6. V-Modell XT Development Stages

acceptance criteria and other subjects. Functional requirements are defined as use cases. The *Requirements Engineer (Supplier)* is responsible for refining the Requirements Specification into an *Overall System Specification*, which is the basis for any system development and documentation activities. The overall system level comprises Integrated Logistic Support (ILS) documentation, enabling systems and the technical system under development, and contains an overall system architecture and the list of system interfaces. We concentrate in the following on the technical system.

The V-Modell XT divides a technical system into system and unit level. The system contains hardware, software and documentation parts which can be developed, acquired or pre-existing. Systems decompose into segments (if necessary) and at the lowest level into units. Units are either pure software, hardware or external. Software/hardware units decompose further into components (if necessary) and modules. The V-Modell XT has a uniform structure for specifying and designing these system elements:

- A *Specification* document describes the context and purpose of a system element and its interfaces from a black-box view, as well as non-functional requirements and internal interfaces between sub-elements.
- An *Architecture* document describes the architecture of one system element and its parts, by documenting architectural principles, design alternatives, decomposition into sub-elements, cross-cutting concerns (such as transaction and security handling), internal interfaces and dependencies, and more.
- An *Implementation, Integration and Evaluation Concept* describes details and plans about the actual implementation with tools and procedures used, the integration and the execution environment, as well as any test and verification strategies.

The uniform structure of system element specification, architecture design and decomposition makes development very systematic, despite the substantially different tasks of system, software, and hardware engineering. Product dependencies ensure that the contents in all products are

consistent, for instance that all requirements are realized by architecture elements. Role responsibilities depend on the level of decomposition: *System Architect*, *Software Architect*, *Hardware Architect*, *Software/Hardware Developer* and *System Integrator* are roles that create the respective products, by performing the activities (with sub activities) associated with the products.

V-Modell XT Extension Mechanism The V-Modell XT can be modularly extended by adding new process modules, which extend a number of existing process modules. A new process module can contain definitions for new work products, activities and roles. New product dependencies to existing products will make sure that the new products are created when required. Seamless extension is possible by adding new subjects, and sub-activities to existing elements.

4. Extending the V-Modell XT

As described above, the V-Modell XT provides a full workflow and process for systems development. It does not, however, give specific methodological guidelines for service-oriented development. We will provide this in the following by describing the integration of our service-oriented development approach into the *V-Modell XT* [4].

We define a new process module "Service-Oriented Systems Development" as shown in Fig. 7. We make use of the fine-grained extension mechanism by adding new subjects to existing work products and new sub-activities to existing activities. Additionally, we add product dependencies to keep our additions consistent with the rest.

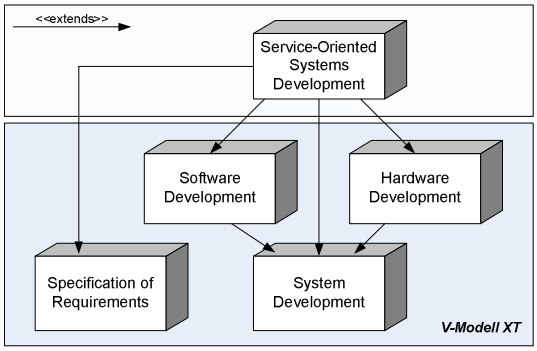


Figure 7. Service-Oriented V-Modell XT Extension

Work products are the main V-Modell XT elements and core project results. Table 2 shows a mapping of the result artifacts of our service-oriented approach to work products and their subjects of the V-Modell XT. V-Modell elements marked with an asterisk (*) are additions. We follow the V-Modell's hierarchical system decomposition into system elements similarly for our service-oriented approach. We define a service model and mapping to an architecture decomposition for each system element with specification and architecture documents. Our service-oriented approach thus scales in the same way as the V-Modell XT.

Element	V-Modell Element	Description
Use Case Graph	Requirements Spec.: *Functional Req's* Overall System Spec: *Functional Req's*	Use Case descriptions are part of the existing requirements documentation.
Services	System/SW/HW Spec.: *Service Access Points** System/SW/HW Arch.: *Service Specifications**	Service interfaces, accessible from outside; service specifications as interaction patterns.
Roles	System/SW/HW Arch.: *Role Model**	The list of roles, their descriptions and the states they can be in.
Role Domain Model	System/SW/HW Arch.: *Role Model**	Description of communication channels between roles.
Compon. Configuration	System/SW/HW Arch.: *Decomposition*	The sub-element structure as known.
Mapping	System/SW/HW Arch.: *Role to Component Mapping**	Maps roles (and thereby service behavior) to sub-elements.
Architecture	System/SW/HW Impl., Int. and Eval. Concept: *Integration Procedures*	The exact deployment configuration as instance network.

Table 2. Product Mapping

We add the following product dependencies that ensure consistency across work products:

- *Consistent Service Refinement*, between products *Overall System Specification* and *Architecture*: the services and roles realize the defined use cases and non-functional requirements.
- *Consistent Service Model*, between products *Specification* and *Architecture*: All roles used within service specifications must be defined; the role structure must reflect the interaction patterns from the service specifications; the service access points must match the service specifications.
- *Consistent Service Mapping*, between products *Specification* and *Architecture*: the sub-element structure as given by the decomposition must be a refinement of the role model; all roles and services must be mapped to sub-elements.

We do not require additional activities; instead we extend existing ones with relevant sub-activities. The sub-activity definitions follow our explanations in Sect. 2:

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

- The V-Modell activities *Preparing System/Software/Hardware Specification* get the new sub-activities *Defining Service Interfaces*.
- The V-Modell activities *Preparing System/Software/Hardware Architecture* get the new sub-activities *Defining Roles and Role Structure*, *Defining System Element Services*, and *Mapping Roles to System Elements*

The integration of our model affects the responsibilities of five existing roles of the V-Modell XT. Their role profile descriptions are sufficiently abstract and fit our purposes. Thus, only slight extensions need to be performed:

- For the roles *Requirements Engineer (Acquirer)* and *Requirements Engineer (Supplier)*, we add domain-modeling and service-oriented design as required capability.
- For the roles *System Architect*, *Software Architect* and *Hardware Architect*, we require knowledge of service-oriented design and of service-oriented architectures and infrastructures.

5. Experiences and Discussion

We have applied the extended V-Modell XT to our BART example in the following way: We start with the *Requirements Specification* containing the system use cases extracted from [41]. We begin system development by refining the requirements into the *Overall System Specification* document, where we specify the BART system as our technical system. The *System Architect* designs the initial *System Architecture* similar to Fig. 1, consisting of the segments Train, Station, ExternalSource, and SafetyComputer, taking the architectural constraints of [41] into account. We analyze the use cases and identify the roles and services as explained in Sect. 2. We list the roles and their connections (see Fig. 2) in the *Role Model* subject and the services (as specified in Fig. 3 and 4) in the *Service Specification* subject of the *System Architecture*. We define the external service access points in the *System Specification*, for instance the communication protocol between Train and Station. With the domain knowledge from the service elicitation steps, the *System Architect* designs the actual *System Decomposition* in the *System Architecture* containing the elements given in Fig. 5. We follow the procedure of Sect. 2 and document the mapping of roles to services in the subject *Role to Component Mapping* of the *Software Architecture*, as shown in Table 1. The actual deployment architecture (see also Fig. 5) is described in the *System Impl., Integration and Eval. Concept*. The system decomposes further into software, hardware and external units (such as the Interlocking System), and into components and modules. We do not describe the detail level specifications and architectures; they follow the standard V-Modell XT development process without service modeling. We perform V-Modell XT activities to edit the work products. Each work product is subject to a Quality Assurance procedure, which includes checks of the product dependencies, including the ones listed above.

Following the above, we have conducted the BART case study as service-oriented V-Modell XT development project. We have shown how the artifacts of our service modeling approach fit in the V-Modell XT work results, in general and for our case example. In summary, we find that our approach blends well with the V-Modell XT. The necessary changes and additions to use the model fit with the existing structure and require only additions and slight extensions of existing V-Modell XT elements. They can all be packaged nicely as a process module. An embedding into other process models with a similar structuring, such as the RUP [12], should be possible with a comparable effort.

One of the challenges of integrating our model-based service-oriented development approach into the V-Modell XT is the V-Modell's strict notion of work products as sole project results. Work products are mostly documents including design model views that describe the system under development. Model-based approaches on the other hand often make use of integrated models, modeling environments and highly iterative modeling cycles. With good tool support, it is possible to keep the service-oriented model consistent across a number of documents, hierarchies and iterations.

Another challenge lies in the structure of the V-Modell XT system elements. The technical system (as part of the overall system) is hierarchically decomposed into segments and these into units, which are either software or hardware. Units have their own specifications and architecture documents and decompose into components and modules. Services in our understanding are cross-cutting across all structural system elements. They are defined as interaction patterns (or protocols) between interacting components. The solution lies in the strategy how the *System Architect* decomposes a system compliant to the V-Modell. The system elements need to be defined such that services can span all required components. New levels of hierarchy are suitable, where services cross-cut only direct subcomponents. This creates a layered service (and system) hierarchy.

6. Related Work

The notion of service and service-oriented development is used in many different application domains and on various levels of abstraction in the Software Engineering community [36, 18]. Its roots lie in the domain of telecommunication systems, where features and their interactions play an important role in software development [37, 28, 29, 42].

Intensive application of service-oriented approaches can be observed for web services [26, 39], web service-oriented architectures [32, 39] and increasingly for embedded automotive systems [21, 17, 3]. Implementation-oriented and infrastructure concerns, including web services [32] and corresponding technologies, such as WSDL [38], WS-BPEL[22], .NET [27] and J2EE [34], or specific mechanisms such as registration, discovery and binding are relevant as members of the deployment technology space for service-oriented development. In the realm of web services standards there has been important work on Web Services Semantics (WSDL-S) [40], Web Services Modeling Ontology (WSMO) [31] and Semantic Markup for Web Services (OWL-S) [25]. OWL-S, in particular, describes both a "service profile" and a "service grounding", which represent "what the service does" and how it maps to underlying messaging protocols and deployment technologies, respectively. To describe the functionality of a service, OWL-S uses a process notion, which uses a limited set of operators to build composite processes – in particular, the notion of joining overlapping services is missing from this approach. The deployment model underlying all of OWL-S and WSDL-S is oriented more towards services as I/O processes, whereas our service notion is closer to the "conversations" or orchestration underlying WS-BPEL [22]; WS-BPEL, however, also would benefit from an operator for disentangling services, such as the join operator we have introduced, above.

Because our approach rests on an end-to-end, interaction-pattern-based service definition (see below), it integrates well with ontologies [25, 31] capturing non-functional aspects – including, but not limited to, security and Quality-of-Service [1].

Our approach is related to the Model-Driven Architecture (MDA) [23], Model-Integrated Computing [35, 10], aspect-oriented modeling (AOM) [7] and architecture-centric software development (ACD) [24]; similar to MDA and ACD we also separate the software architecture into abstract and concrete models. In essence, the service elicitation and architecture definition phases in our development process correspond to building "platform-independent models" (PIMs) and "platform dependent models" (PDMs), respectively. In contrast to MDA and ACD, however, we consider services and their defining interaction patterns as first-class, cross-cutting modeling elements of both the abstract and the concrete models. This also distinguishes our approach from Model-Integrated Computing and AOM.

Ambler uses process patterns in [2] to describe task-specific self-contained pieces of processes and workflows in a reusable way. Such patterns can be applied to solve complex tasks when needed. Störrle [33] shows how process patterns can be described in great detail using UML. The idea of process patterns is further refined by Gnatz et al. [8] in the form of a modular and extensible software development process based on collections of independent process components. These process patterns essentially are the basis of the extension mechanism of the V-Modell XT.

7. Summary and Outlook

Service-oriented development promises to address many complexities in the development of ultra-large scale (ULS) systems. We have explained our service-oriented approach and corresponding notations using the BART system case study. Our approach seems to be well suited to managing the complexity of this distributed, reactive system, by focusing on services as first-class entities throughout the development process.

We have integrated our service-oriented development approach into an existing systems development process model by describing a service-oriented extension of the V-Modell XT. We have shown how our approach can be realized through extensions of V-Modell activities and product definitions. We ensure consistency to existing V-Modell concepts by introducing product dependencies. The V-Modell XT extension appears to be seamless and intuitive. We have enriched the flexible organization and management framework with valuable methodical detail for service-oriented development. Thus, we support the development of ULS systems by suitable models and notations as well as by a large-scale systematic development process.

Future work will include conducting a larger case study or V-Modell XT pilot project with the service-oriented extensions. We would also like to refine our extensions to include non-functional requirements and quality properties.

References

[1] J. Ahluwalia, I. Krüger, M. Meisinger, and W. Phillips. Model-Based Run-Time Monitoring of End-to-End Deadlines. In *Proceedings of the Conference on Embedded Systems Software (EMSOFT)*, 2005.

[2] S. W. Ambler. *More Process Patterns: Delivering Large-Scale Systems Using Object Technology*. Cambridge University Press, 1999.

[3] M. Broy, I. H. Krüger, and M. Meisinger, editors. *Automotive Software - Connected Services in Mobile Networks. Proceedings of the Automotive Software Workshop San Diego 2004*. Lecture Notes in Computer Science, Volume 4147, Springer, New York, 2006.

[4] Bundesrepublik Deutschland. *V-Modell XT*, 1.2 edition, 2006. `http://www.v-modell-xt.de/`.

[5] V. Ermagan, T.-J. Huang, I. Krüger, M. Meisinger, M. Menarini, and P. Moorthy. Towards Tool Support for Service-Oriented Development of Embedded Automotive Systems. In *Proceedings of the Dagstuhl Workshop on Model-Based Development of Embedded Systems (MBEES'07)*, To appear as LNCS. Springer Verlag, 2007.

[6] E. Evans. *Domain Driven Design*. Addison-Wesley, 2003.

[7] G. Georg, R. France, and I. Ray. Composing Aspect Models. In *Proceedings of the 4th AOSD Modeling With UML Workshop*, 2003.

[8] M. Gnatz, F. Marschall, G. Popp, A. Rausch, and W. Schwerin. The living software development process. *Software Quality Professional*, 5(3), June 2003.

[9] ITU-TS. Recommendation Z.120: Message Sequence Chart (MSC). Geneva, 1996.

[10] G. Karsai, J. Sztipanovits, A. Ledeczi, and T. Bapty. Model-Integrated Development of Embedded Software. In *Proceedings of IEEE January 2003*, 2003.

[11] F. Kordon and L. Michel, editors. *Formal Methods for Embedded Distributed Systems*. Springer, 2004.

[12] P. Kruchten. *The Rational Unified Process. An Introduction*. Addison-Wesley, 2nd edition, 2000.

[13] I. Krüger. *Distributed System Design with Message Sequence Charts*. PhD thesis, Technische Universität München, 2000.

[14] I. Krüger, R. Mathew, and M. Meisinger. Efficient Exploration of Service-Oriented Architectures using Aspects. In *Proceedings of the 28th International Conference on Software Engineering (ICSE)*, 2006.

[15] I. Krüger, M. Meisinger, and M. Menarini. Applying Service-Oriented Development to a Complex System: the BART case study. In *Proceedings of the Monterey Workshop 2005*, number 4322 in LNCS. Springer Verlag, 2007.

[16] I. Krüger, M. Meisinger, M. Menarini, and S. Pasco. Rapid Systems of Systems Integration - Combining an Architecture-Centric Approach with Enterprise Service Bus Infrastructure. In *Proceedings of the 2006 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 51–56. IEEE, 2006.

[17] I. Krüger, E. C. Nelson, and V. Prasad. Service-based Software Development for Automotive Applications. In *CONVERGENCE 2004*, 2004.

[18] I. Krüger, B. Schätz, M. Broy, and H. Hussmann, editors. *SBSE'03. Service-Based Software Engineering. Proceedings of the FM2003 Workshop*, number TUM-I0315, 2003.

[19] I. H. Krüger. Capturing Overlapping, Triggered, and Preemptive Collaborations Using MSCs. In M. Pezzè, editor, *FASE 2003*, volume 2621 of *LNCS*, pages 387–402. Springer Verlag, 2003.

[20] I. H. Krüger, R. Mathew, S. Leue, and T. Systä. Component Synthesis from Service Specifications. In *Scenarios: Models, Transformations and Tools*, volume 3466 of *LNCS*, pages 255–277. Springer Verlag, 2005.

[21] E. C. Nelson and K. V. Prasaad. Automotive Infotronics: An emerging domain for Service-Based Architecture. In I. H. Krüger, B. Schätz, M. Broy, and H. Hussmann, editors, *SBSE'03 Service-Based Software Engineering, Proceedings of the FM2003 Workshop*, Technical Report TUM-I0315, pages 3–14. Technische Universität München, 2003.

[22] OASIS. Web Services Business Process Execution Language (WS-BPEL), Version 2.0. Specification public draft, 23-Aug-2006, 2006. `http://docs.oasis-open.org/wsbpel/2.0/wsbpel-specification-draft.pdf`.

[23] OMG (Object Management Group). Model Driven Architecture (MDA). MDA Guide 1.0.1, omg/03-06-01, 2003. `http://www.omg.org/mda`.

[24] OMG (Object Management Group). UML, Version 2.0. OMG Specification formal/05-07-04 (superstructure) and formal/05-07-05 (infrastructure), 2005.

[25] W3C: OWL-S: Semantic Markup for Web Services, W3C Member Submission 22 November 2004, 2004. `http://www.w3.org/Submission/OWL-S/`.

[26] C. Peltz. Web Services Orchestration and Choreography. *IEEE Computer*, 36(10):46–52, 2003.

[27] D. S. Platt and K. Ballinger. *Introducing Microsoft .NET*. Microsoft Press, 2001.

[28] C. Prehofer. Feature Oriented Programming: A fresh look at objects. In *Proceedings of ECOOP 1997*, number 1241 in LNCS. Springer Verlag, 1997.

[29] C. Prehofer. Plug-and-Play Composition of Features and Feature Interactions with State-chart Diagrams. In *Proceedings of the 7th International Workshop on Feature Interactions in Telecommunications and Software Systems*, 2003.

[30] A. Rausch and M. Broy. *Das V-Modell XT. Grundlagen, Erfahrungen und Werkzeuge*. dpunkt.verlag, 2006. In German.

[31] D. Roman, U. Keller, H. Lausen, R. L. J. de Bruijn, M. Stollberg, A. Polleres, C. Feier, C. Bussler, and D. Fensel. Web service modeling ontology. *Applied Ontology*, 1(1):77–106, 2005.

[32] J. Snell, D. Tidwell, and P. Kulchenko. *Programming Web Services with SOAP*. O'Reilly, 2002.

[33] H. Störrle. Describing Process Patterns with UML. In *Proc. of the 8th European Workshop on Software Process Technology (EWSPT '01)*, pages 173–182. Springer, 2001.

[34] SUN Microsystems Inc. Java Platform, Enterprise Edition (Java EE, J2EE). Website, 15-Nov 2006. `http://java.sun.com/javaee/`.

[35] J. Sztipanovits and G. Karsai. Model-Integrated Computing. *IEEE Computer*, 30(4):110–112, April 1997.

[36] D. Trowbridge, U. Roxburgh, G. Hohpe, D. Manolescu, and E. Nadhan. *Integration Patterns. Patterns & Practices*. Microsoft Press, 2004.

[37] K. J. Turner. Relating Services and Features in the Intelligent Network. In *Proc. of the 4th International Conference on Telecommunications*, pages 235–243, 1997.

[38] W3C. Web Services Description Language (WSDL), Version 1.1. W3C Note, 15-Mar-2001, 2001. `http://www.w3.org/TR/wsdl/`.

[39] W3C. Web Services Architecture, 11-Feb 2004. `http://www.w3.org/TR/ws-arch/`.

[40] W3C: Web Service Semantics - WSDL-S, Version 1.0. W3C Member Submission, 7-Nov-2005, 2005. `http://www.w3.org/Submission/WSDL-S/`.

[41] V. Winter, F. Kordon, and L. Michel. The BART Case Study. In F. Kordon and L. Michel, editors, *Formal Methods for Embedded Distributed Systems*, pages 3–22. Springer, 2004.

[42] P. Zave. Feature-Oriented Description, Formal Methods, and DFC. In *Proceedings of the FIREworks Workshop on Language Constructs for Describing Features*, pages 11–26. Springer-Verlag, 2001.

Selecting Requirements Engineering Techniques based on Project Attributes - A Case Study

[1]Li Jiang [2]Armin Eberlein

[1] *School of Computer Science, The University of Adelaide, SA, 5000, Australia*

[2] *Computer Engineering Department, American University of Sharjah, UAE*

Abstract

Selection of the most appropriate RE techniques for a software project based on the project's characteristics is a non-trivial process and a common challenge faced by software developers. In order to facilitate RE techniques selection, we propose a model for RE Technique Suitability Assessment (RETSA) based on project attributes. This model was developed based on surveys and interviews of experienced software developers and experts from industry and academia. RETSA was applied to an industrial project. This case study shows the help this model provided during the selection of RE techniques for a software project.

Keywords: Requirements engineering, requirements techniques selection, software project attributes, assessment.

1. Introduction

As one of the processes in software engineering, requirements engineering (RE) plays a vital role in ensuring the overall quality of software products and the success of software projects [1-5]. However, research has shown that many of the key problems that the software industry faces today are still the same as the ones experienced about 40 years ago [6-8]. Poor RE practices have been blamed as one of the major reasons contributing to the ongoing problems. As has been emphasized by many researchers, the RE process is fairly complex as it involves many disciplines such as sociology, economy, system engineering, computer science, management, psychology [1, 9]. So far it has been impossible to find one technique that fits all problem domains [2, 3, 10].

For a non-trivial software project, several techniques are needed in the RE process [11-13]. Numerous techniques have been developed in the last three decades which aim at providing support for RE processes, yet in reality, there is still a big gap between theory and practice. One of the major reasons for this is the lack of support for the selection of the most suitable RE techniques for a specific software project. The selection of RE techniques is still largely based on personal preference rather than on characteristics of the project at hand. So far, only a limited amount of research has been done into the selection of RE techniques. The few approaches that currently exist provide only little guidance for the selection process. Effective ways have to be found to fill this gap.

The objectives of this research were, therefore, to investigate the feasibility of the RETSA model to provide decision support for the selection of RE techniques and then to assess the suitability of those techniques for a particular software project.

The selection of RE techniques involves many factors that need to be considered in a systematic way. We have investigated three ways to facilitate the overall process:

(1) Development of an RE techniques knowledge base. We identified 46 well-documented RE techniques [10, 15] and included them in this knowledge base. An evaluation schema was developed based on a detailed analysis of RE techniques using various analytic methods, such as clustering, comparison, etc [16].

(2) Construction of a knowledge base of application experiences of RE techniques. This knowledge base includes a collection of software projects and past experiences of using RE techniques in these projects [14]

(3) Evaluation of the suitability of RE techniques for software projects using project characteristics. The experience and knowledge of software experts were very important for the development of this RE techniques suitability Assessment (RETSA) Model, which is one of the models used in our Methodology for RE Technique Selection (MRETS) [17].

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

The aim of the RETSA model is to provide support for selecting RE techniques for the whole RE process, including requirements elicitation, requirements analysis and negation, requirements documentation, and requirements verification and validation [1]

The rest of this paper is organized as follows: The RETSA model is presented in Section 2. A case study in which RETSA was used to support RE technique selection is reported in Section 3. Conclusion and further work are discussed in Section 4.

Requirements Volatility, Project Category, Degree of Safety Criticality, Time Constraints, Cost Constraints. Detailed definitions of all of these attributes can be found in [10].

It has to be mentioned that these attributes are considered important factors as their values determine essential characteristics of a software project [18; 20; 22; 23]. At the beginning of using the RETSA model to support RE techniques selection for a software project, the above attributes have to be estimated based on the experience of requirements engineers. This estimation provides useful information and a better understanding of the software project initially. Additionally the information provides a foundation for the decision making process for selection of suitable RE techniques which will help to derive quality requirements through the RE process.

Table 1 Project attributes that influence RE techniques selection

Project attribues	Reasons for the definition of the attributes	Definition and description of project attribute (Refer to [10] for more information about the definitions of the attributes)
Project Size	The size of project is important to the RE techniques selection. For example, large projects require systematic techniques to elicit, analyze, document, verify and validate requirements.	This attribute is defined as the size X of a project in terms of number of requirements. The requirements refer to atomic requirements which are defined as indivisible "well-formed" requirements [24]. Possible values for this attribute are: very small (X<100 requirements), small (100<=X<500), medium (500<=X<1000), big (1000<=X<4000), very big (X>=4000).
Project Complexity	The complexity of a project matters to the selection of RE techniques. For example, a project with high complexity requires systematic techniques to be used in the RE process.	The complexity of a project is estimated using factors such as the number of project requirements, the overall system architecture, the relationships between requirements and the heterogeneity of stakeholders. The value of the complexity of a project is defined as very low, low, medium, high, and very high.
Requirements Volatility	Requirements volatility is an important attribute that has to be considered in the selection of RE techniques. For example, projects with higher requirements volatility require more flexible techniques to be used in a RE process.	This attribute is defined as the percentage Y of requirements that change throughout the development of the project. The attribute can have the following values: very low (Y<1%), low (10%>Y>=1%), average (30%>Y>=10%), high (50%>Y>=30%), and very high (Y>=50%).
Project Category	Projects in different categories require different techniques to be used in the RE process. For example, the techniques used in a safety-critical system will not be the same as the ones used in a non-safety critical system.	This attribute defines the type of project. Possible values are: Communication, Embedded, Semi-detached and Organic. Some of these values are borrowed from the COCOMO model [25].
Degree of Safety Criticality	Degree of safety criticality is considered as an important attribute for the selection of RE techniques. Projects with a high degree of safety criticality require more rigorous and disciplined techniques.	This attribute is defined as the degree of safety required by the system measured by the potential loss of human life or property. The values are defined as follows: very low, low, medium, high, and very high.
Time Constraints	Time constraints have to be considered when it comes to the selection of RE techniques. Projects with high time constraints require lightweight techniques to be used because heavy-weight techniques will significantly delay the overall project.	Time constraints are defined as the degree of the time-to-market pressure for the software project. The attribute values are defined as: very low, low, medium, high, and very high.
Cost Constraints	Projects with high cost constraints require lightweight techniques to be used in the RE process because heavy-weight techniques will increase the cost, especially when training for the use of the technique is required.	Cost constraints are defined as the ratio of the overall budget of the project with respect to its actual cost. The attribute values are defined as: very low, low, medium, high, and very high.

2. Technique Suitability Model

The selection of RE techniques for a specific project requires the assessment of RE techniques with respect to software project attributes. To this end, research into RE techniques through analysis, synthesis, and classification mechanisms was carried out in this research.

Based on literature [18-21], we defined 21 attributes of software projects [10]. However, for simplification, we focus in this paper on the evaluation of RE techniques based on only 7 project attributes. This treatment is reasonable since the objective of the research is to demonstrate the usefulness of the proposed RETSA model. The seven attributes are (see Table 1 for detailed definitions): Project Size, Project Complexity,

Based on our research into 46 well-documented RE techniques [10] and existing research in RE techniques [19-22], a Technique Suitability Assessment Matrix was developed. This matrix contains information about the suitability of the identified techniques with respect to the seven attributes of software projects mentioned above.

After derivation of the Technique Suitability Assessment Matrix, surveys and interviews of experts were conducted to examine and validate the Technique Suitability Assessment Matrix.

19 developers completed the survey. All of them had 3 to 10 years industrial experience. The two industry experts, who participated in the interviews, had more than 10 years of work experience and were involved in more than 27 and 32 software projects. Three experts came from academia with more than 10 years experience in RE research. Any disagreements during the validation of the Technique Suitability Assessment Matrix were resolved through discussions. When no consensus could be reached, the weighted average aggregation method [26, 27] was used to determine the appropriate values of the Technique Suitability Assessment Matrix. A higher weight was given to industry experts and developers to ensure industrial relevance of our work.

Table 2 An Example of Technique Suitability Assessment Matrix

Attributes of Project		Unified Modeling Language(UML)	SDL	User Story Card
Project Size	Very big	1	0.5	0
	Big	1	0.75	0.25
	Medium	1	1	0.5
	small	0.75	0.75	0.75
	very small	0.5	0.5	1
Project Complexity	Very High	1	0.5	0
	High	1	0.75	0.25
	Medium	1	1	0.5
	Low	0.75	0.75	0.75
	Very Low	0.5	0.5	1
Requirements Volatility	Very High	0.5	0	1
	High	0.75	0.25	0.75
	Medium	1	0.5	0.5
	Low	1	0.75	0.25
	Very Low	1	1	0

Notes: Specification Description Language (SDL), Unified Modeling Language (UML) and User Story Card are three requirements documentation techniques.

The final version of the Technique Suitability Assessment Matrix served as the experience library to help select RE techniques based on the characteristics of the software project.

An example of the Technique Suitability Assessment Matrix for some RE techniques is shown in Table 2. As can be seen in this table, the suitability of the technique "Unified Modeling Language (UML)" is very high (normalized value 1) when the project size is very big, big and medium; the suitability of the technique is medium (normalized value is 0.5) when the project size is very small.

More details on the current data in the Technique Suitability Assessment Matrix can be found in [10].

The Technique Suitability Assessment Matrix shows the suitability of techniques for a software project based on the project's characteristics. This matrix can be represented as the following function:

$F: \ T \times A \times V \rightarrow R'$, $\quad R' = [0, 1]$

where:

T: represents all current techniques in the Suitability Assessment Matrix,

$T=\{t_1, \ldots, t_m\}$, t_i represents a specific RE technique, while m is the number of techniques.

A: represents project attributes; $A = \{A_1, \ldots, A_n\}$, n is the number of attributes used in the assessment model ($n=7$ in the current Suitability Assessment Matrix).

V: represents possible values of project attributes; $V=\{v_{1,1}, \ldots, v_{n,k}\}$, k is the number of values for each attribute A_j.

A tuple $(A_j, v_{j\,q})$ represents a specific situation of a project attribute A_j with value $v_{j,q}$.

In most cases, a specific software project situation is determined by a set of such tuples, i.e. $\{(A_1, v_{1,1}), \ldots, (A_n, v_{n,k})\}$.

With these definitions, a specific software project $\mathbf{P_s}$ can be represented as a subset of A × V, i.e. for any software project $\mathbf{P_s} \subset A \times V$.

For a technique i, denoted as t_i, the suitability S_{t_i} of a technique for a given project can be calculated as:

$$S_{t_i} = \frac{\sum_j W_{A_j} * F\ (t_i, A_j, v_{j,q})}{n},$$

$$j=[1..n],\ q=[1..k],\ \ W_{A_j}=[0,\ 0.2,\ 0.4,\ 0.6,\ 0.8,\ 1] \qquad (2.1)$$

where n is the number of all attributes used in the evaluation. W_{A_j} is the weight given to each attribute by the requirements engineers, so that each attribute is adequately considered.

If a set of techniques $T=\{t_1, \ldots, t_p\}$ is selected for the process model for project P_s, the overall suitability S_T of the process model for project P_s can be calculated as:

$$S_T = \frac{\sum_i \frac{W_{t_i} * \sum_j W_{A_j} * F\ (t_i, A_j, v_{j,q})}{n}}{p},$$

$$i=[1..p],\ j=[1..n],\ q=[1..k],\ W_{A_j}=[0..1],\ W_{t_i}=[0..1], \quad (2.2)$$

where n is the number of attributes used in the evaluation and p is the number of techniques in T. W_{t_i} is the familiarity factor given to each technique based on the familiarity of the RE team with the technique. The reason for using the familiarity factor W_{t_i} is to consider the likely cost associated with the necessary training of the team for using the RE techniques. The value of W_{t_i} for a technique t is low if the RE team does not know the technique t.

An example of using this model to evaluate the suitability of a technique for a certain software project is illustrated

in the following. For simplicity, we assumed that Project Size, Project Complexity, and Requirements Volatility are the major concerns for selection of the RE techniques; therefore, the suitability of the techniques are only assessed with the three project attributes given above and the techniques selection was only carried out for the requirements documentation stage. Nevertheless, the RETSA model can be used for the entire RE process(see Section 3 for information).

A software project is assumed to have the following attributes:

Project Size=Big
Project Complexity=High
Requirements Volatility=Low

Let's assume that there are three process models. Each one of these models uses a different requirements documentation technique for the software project. Our Suitability Assessment Model RETSA is then used to determine which one of the three process models uses the best documentation technique. More information about the process model options, RE techniques assessment with respect to the attributes of the project, and the calculation of the overall suitability of each RE process model is summarized in Table 3 (assuming $W_{A_j}=1$, $W_{t_i}=1$ for all i and j in this example for simplicity):

Based on the calculations (see Table 3) according to the RETSA model, we can say that the technique UML is the most suitable one of the three techniques based on the given software project attributes. This means that the process model P_1, which uses UML, is the most suitable process model based on the given characteristics of the software project according to our RETSA model. Thus, the process model P_1 is recommended. RE process models that use more than one technique are assessed using the same principles. However, the results are combined using formula (2) to get the overall suitability of a RE process model for a project.

Further research in technique suitability assessment is still needed. The limitations of our current research are that the number of RE techniques included in the suitability assessment matrix is still small, and more characteristics of the software project need to be taken into consideration in order to provide more information for RE techniques selection. However, this research is an initial effort towards the assessment of the suitability of RE techniques for a specific software project. So far, the assessment model has been very useful for techniques selection in the case studies carried out one of which is presented in Section 3.

It has to be mentioned that ideally, the RETSA model can be used for all types of software projects. However, it fits best to projects that use a complete RE process. In the current version of RETSA, support for the selection of RE techniques for Agile process and COTS-based software development is still weak and subject to further research. Additionally, tailoring and customizing existing techniques to specific projects has not yet been covered.

Table 3 Process options and suitability calculation

RE Process Options / RE Techniques assessment with respect to the project attributes / Project situation and suitability calculation		*Process P_1* (UML is suggested)	*Process P_2* (SDL is suggested)	*Process P_3* (User Story Card is suggested)
Project situation (A_j, v_j) $j=1..3$	Project Size= Big	1	0.75	0.25
	Project Complexity= High	1	0.75	0.25
	Requirements Volatility=Low	1	0.75	0.25
Suitability calculation	$S_{\mathrm{UML}}=\frac{\sum_j F(\mathrm{UML}, A_j, v_{j,q})}{3}$	1		
	$S_{\mathrm{SDL}}=\frac{\sum_j F(\mathrm{SDL}, A_j, v_{j,q})}{3}$		0.75	
	$S_{\mathrm{UserStoryCard}}=\frac{\sum_j F(\mathrm{UserStoryCard}, A_j, v_{j,q})}{3}$			0.25

3. A Case Study

The case study presented in this paper is part of a larger study conducted during our research to examine the merits of a Methodology for Requirements Engineering Techniques Selection (MRETS) [10]. MRETS includes several analytic and assessment models which are used to assess and analyze RE techniques from different perspectives. The objective of MRETS is to help requirements engineers select the most suitable RE techniques for a project.

For simplicity, we have limited our presentation in this paper to that portion of the overall case study where the proposed RETSA model is relevant. This portion of the case study addresses the question: How can the Technique Suitability Assessment model RETSA help assess the suitability of the candidate RE techniques and help select the most suitable RE techniques? To answer this question, we formed the following hypothesis:

The Technique Suitability Assessment model RETSA can help select RE techniques that match the characteristics of the software project and can help produce high-quality requirements in a shorter period of time.

To validate this hypothesis, we have to examine if the RE techniques selected with the help of RETSA

contribute to the on-time delivery of the software project, and the satisfaction of management and software developers. Unlike formal experiments, we acknowledge that case studies do not have a well-defined theoretical basis from which one can draw strong conclusions. However, it is widely believed in the software engineering domain that real-life case studies are suitable for an industrial evaluation of software engineering techniques and tools if they are conducted in a sound way [28].

In general, the case study followed the following steps [29]:

(1) Definition of hypothesis
(2) Selection of the pilot project
(3) Selection of a suitable method for comparison and criteria for the validation
(4) Consideration of the effects of confounding factors
(5) Planning of the case study
(6) Conducting and monitoring of the case study against the plan
(7) Analysis of results and generation of report

For simplicity, we only summarize the case study in the following two sub-sections which include software project introduction, RE techniques selection with the help of RETSA, case study results and their analysis. Brief conclusions, including the advantages and limitations of the case study, are presented at the end of this paper.

We acknowledge that there are various other factors that contribute to the success of an RE process, such as the knowledge of requirements engineers and management commitment. Additionally, there are numerous variables that need to be measured and controlled during the case study. For space reasons, we will focus on the positive and negative effects that the selected RE techniques had on the software project.

3.1 Description of the Software Project

In this case study, a software project called Port Scheduling System (PSS) in Company X (the name of the company is withheld for reasons of confidentiality) was selected and carried out. Company X is a CMM level 2 certified company and its management is committed to software process improvement, especially as RE processes had been ad hoc in most previous software projects. Brief information about the PSS project is given in Table 4.

Table 4 Project Definition

Project Description	This project is to develop a "Port Scheduling System". The objective of the system is to schedule a container terminal with a throughput of maximum 1 million TEU (twenty foot equivalent unit) each year. The terminal must also be able to hand the smaller cargos. The project requires a highly interactive interface……
Project Attributes	**Project Size**: Medium (>=500 and <1000 Requirements)
	Project Complexity: Medium
	Requirements Volatility: Low
	Project Category: Semi-Detached
	Degree of Safety Criticality: High
	Cost Constraint: Low
	Cost Constraint: Medium

Table 5 RE Techniques Suitability Value With Respect To The Characteristics of Software Project

RE Techniques that are selected as candidate / Project Attributes and Weight	Ethnography (W_t=0.2)	Focus Group (W_t=1)	Interview (W_t=1)	JAD (W_t=0.8)	Scenario-Based Analysis (W_t=1)	OO Analysis (W_t=1)	AHP (W_t=0.8)	Goal Oriented Analysis (W_t=0.2)	Viewpoints Oriented Analysis (W_t=1)	Structured Natural Language Specification (W_t=1)	SDL (W_t=0.4)	Viewpoints Based Definition (W_t=1)	Formal Requirements Inspection (W_t=1)	Viewpoints Based Validation (W_t=1)
Project Size Value: Medium (W_A=0.8)	0.75	0.75	1	0.75	1	1	1	1	1	1	1	1	1	0.75
Project Complexity Value: Medium (W_A=1)	0.75	0.75	1	1	1	1	1	1	1	1	1	1	1	1
Requirements Volatility Value: Low (W_A=1)	1	1	1	1	1	1	1	1	1	1	0.75	1	1	0.75
Project Category Value: Semi-detached mode: (W_A=1)	1	1	1	1	0.5	0.5	1	1	0.75	1	1	1	1	1
Degree of Safety Criticality Value: Medium (W_A=1)	0.75	0.75	1	1	0.5	0.5	1	1	1	0.5	0.75	0.75	0.75	0.5
Time Constraints Value: Low (W_A=1)	0.75	1	1	1	0.5	1	1	1	1	1	0.75	0.75	0.75	0.75
Cost Constraints Value: Medium (W_A=1)	0.75	0.75	1	1	1	0.75	1	1	1	1	0.75	1	1	0.5

3.2 Selection of RE Techniques

Based on the project attributes and with the help of other mechanisms of the MRETS methodology (such as case based reasoning, RE techniques analysis, calculation of the objective function [10]), a set of candidate RE techniques were recommended which are shown in the first row of Table 5. Based on the RE technique analysis, several sets of RE techniques (see Table 6) were derived and recommended for use in the project. These combinations of RE techniques served as the candidate RE techniques for the software project and were subjected to the final assessment by requirements engineers with respect to their suitability for the project based on its attributes.

Table 6 RE Techniques Combination and Suitability Calculation

No.	Technique Combination	Name of the Techniques	Suitability Calculation S_T
1	T_1	Interview, Focus Group, Ethnography Viewpoint-Based Analysis, AHP Structured Natural Language Specification Viewpoint-Based Validation	4.69
2	T_2	Interview, Focus Group, Ethnography OO Analysis, AHP Viewpoint-Based Definition Formal Requirements Inspection	4.58
3	T_3	Interview, Focus Group, Ethnography Scenario-Based Analysis, AHP Structured Natural Language Specification Formal Requirements Inspection	4.49
4	T_4	Interview, Focus Group, Ethnography Viewpoint-Based Analysis, AHP Viewpoint-Based Definition Formal Requirements Inspection	4.70
5	T_5	Interview, Focus Group, Ethnography Goal-oriented Analysis, AHP Viewpoint-Based Definition Formal Requirements Inspection	4.05
6	T_6	Interview, Ethnography Scenario-Based Analysis, AHP Viewpoint-Based Definition Viewpoint-Based Validation	4.31
7	T_7	JAD, Interview, Ethnography Viewpoint-Based Analysis, AHP Viewpoint-Based Definition Formal Requirements Inspection	4.66
8	T_8	JAD, Interview, Ethnography Scenario-Based Analysis Structured Natural Language Specification Viewpoint-Based Validation	4.28

We used the RETSA model to help us in the decision making process. The data necessary to assess the suitability of each RE technique for the PSS project (shown in Table 5) was retrieved from the RE techniques knowledge library discussed in Section 2. The weight of each attribute W_{A_j} and familiarity factor of the requirements engineers with each technique W_{t_i} are also given in Table 5. Combinations of RE techniques are presented in the third column of Table 6; and the final calculation result of the overall suitability of each combination of RE techniques (S_T) is shown in the fourth column of Table 6. As can be seen in this table, the RE technique combination 4 is considered the most suitable as it has the highest value of S_T .

It is worth mentioning that the values of the overall suitability calculation of RE technique combination T_1 , T_2 , T_7 are quite similar to that of T_4. The decision to select T_4 for the PSS project is not solely based on this calculation. The evaluation of RE techniques attributes with respect to other attributes of the PSS project and the experiences of requirements engineer were also considered during the selection of RE techniques. These aspects are considered by the MRETS methodology [10].
In this case study, the selected techniques combination (T_4) was modified based on the perception of requirements engineers. It was found that the prioritization of requirements in this project was not considered to be that difficult and can be done informally; therefore, AHP was removed during the final review. Thus the final set of RE techniques that were applied to the PSS project was:

Interview, Focus Group, Ethnography, Analysis techniques, Viewpoint-Based Analysis, Viewpoint-Based Definition, and Formal Requirements Inspection

3.3 Results Analysis

The data collected throughout the execution of the case study is presented in Figure 1. The data was compared with a previous project of an Intelligent Industrial Waste-Water Treatment System (IWTS) that did not use our RETSA assessment model. Both projects exhibited similar project attributes. Figure 1 compares the two projects carried out by the same team except that two junior developers were not part of the PSS project. The two projects are similar in the number of people involved and the planned duration. However, even though the PSS project had about 25% more requirements than the IWTS project, it required less development time. Furthermore, the PSS project was only 8.33% overtime, while the IWTS project was 31.25% overtime in terms of person-months. The likely reason for this is that the PSS RE Process helped discover and correct more requirements earlier on than was the case in the IWTS project, which experienced very late requirements changes. A key success indicator is that no major requirement that would have had a significant impact on the overall system structure or on major functionality was added or deleted after the completion of the requirements engineering process.

In summary, the comparative evaluation of the two projects shows the advantages of using the proposed RETSA model in helping select the most suitable RE techniques.

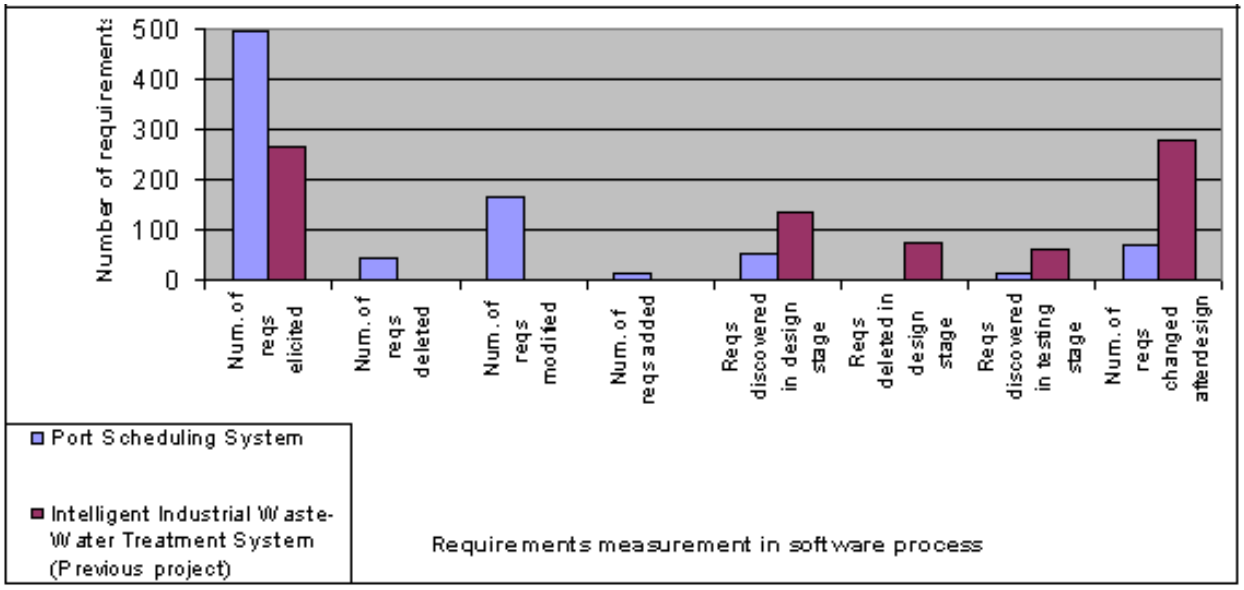


Fig. 1 Comparison of the Measurements between the PPS and IWTS Projects

Table 7 Basic Survey Information

Subjects	Number of Questionnaire Issued	Number of Respondents with Valid Answers
Managers (include management staff directly involved in the project)	4	3
Developers	49	42
Requirements engineers (including analyst, architects)	4	4

In addition to the quantitative analysis, a questionnaire was completed by the developers, requirements engineers as well as managers who were involved in the PSS project. The objective of the survey was to get further feedback about the usage of the RETSA model as well as MRETS. The basic survey information, survey questions and results derived and analyzed are shown in Table 7, Figures 2, 3 and 4 respectively.

As can be seen from Figure 2, management was very positive regarding the overall performance of the RE process used in the PSS project. The same is valid for all requirements engineers (see Figure 3). This is an indication that the requirements engineers found the techniques selected by RETSA suitable for the PSS project.

Figure 4 shows that some developers chose “Disagree” or “Strongly disagree”. One likely reason is that the PSS RE Process had very little impact on those aspects of the software process in which these developers were involved.

	Strongly Agree	Agree	Medium	Disgree	Strongly Disgree
The RE process model is really the most suitable RE process for project PSS	1	1	1	0	0
The RE process model helps to reduce rework for the overall project	1	2	0	0	0
The RE process model helps to reduce the overall delay of the software project	1	2	0	0	0
The overall satisfaction of the customers regarding the final requirements specification is high	0	3	0	0	0
The RE process is better organized than before	1	2	0	0	0

Note: The numbers in the circle indicate the number of people who selected the corresponding answer

Fig. 2 Survey results for managers

	Strongly Agree	Agree	Medium	Disgree	Strongly Disgree
The RE process addresses the major issues of PSS project	2	2	0	0	0
The notations are suitable for modelling the requirements	1	3	0	0	0
Some other notations such as UML will be very helpful for the modelling	0	2	1	1	0
It will be more productive if some more powerful tool such as Rational Rose and Rational Pro are used for requirements documentation and modelling	0	2	2	0	0
The difficulty and cost to apply a new technique is not very high if appropriate training is available even though it has never been used before.	1	3	0	0	0
The RE process is more organized than before	1	2	1	0	0
Overall the framework FRERE is very helpful for developing the most suitable project model for a particular project	0	3	1	0	0

Note: The numbers in the circle indicate the number of people who selected the corresponding answer

Fig. 3 Survey results for requirements engineers

For example, a developer whose major role is related to the deployment has generally less experiences of the major impact of requirements change. Nevertheless, the average percentage of "Disagree" and "Strongly disagree" combined is only about 12%; while more than 66% of the respondents chose "Strongly agree" and "Agree". If we also include the number of respondents that selected the answer "Medium", then 88% of the developers agree that the RE Process was beneficial for the PSS project. This information serves as further evidence of the usefulness of RETSA

	Strongly Agree	Agree	Medium	Disgree	Strongly Disgree
The RE process addresses the major issues of PSS project	2	2	0	0	0
The notations are suitable for modelling the requirements	1	3	0	0	0
Some other notations such as UML will be very helpful for the modelling	0	2	1	1	0
It will be more productive if some more powerful tool such as Rational Rose and Rational Pro are used for requirements documentation and modelling	0	2	2	0	0
The difficulty and cost to apply a new technique is not very high if appropriate training is available even though it has never been used before.	1	3	0	0	0
The RE process is more organized than before	1	2	1	0	0
Overall the framework FRERE is very helpful for developing the most suitable project model for a particular project	0	3	1	0	0

Note: The numbers in the circle indicate the number of people who selected the corresponding answer

Fig. 4 Survey results **for developers**

3.4 Discussion

The quantitative and qualitative analysis presented above shows that company X was able to develop a much better requirements specification with more precise definitions, clearer structure and traceability by using the RE techniques recommended by RETSA than was the case in the IWTS project which used an ad-hoc RE process. The requirements engineers and project manager emphasized that the high quality of the requirements specification had a positive impact on the software project. The data collected from the case study in the company show that the requirements volatility was lower compared to the IWTS project and requirements ambiguity and conflicts were greatly reduced.

We acknowledge that comparing the data from the two projects cannot be used as formal proof that our RETSA model will always result in the selection of the best RE techniques for a software project. The following factors limit the validity of the case study:

- Management commitment: The management of the two projects had slightly different levels of commitment to the RE process. Management in the PSS project provided good support for using the RE process and techniques recommended in the case study. However, this was not considered a major reason for the good results of the PSS project.
- Learning effects and training: Learning effects play a role if the projects are in the same domain. However, since the two projects are in two different domains, the learning effects were considered minimal. Therefore, these factors shall not be considered as a major reason for the success of the PSS project.
- Available project data: The data collected from the sister project IWTS was limited which did not allow us to compare the two projects as thoroughly as we would have liked. The difficulty in getting sufficient data to verify new theories and arguments in software engineering research has already been identified by Glass [30]. This is also the case in our research.
- Other factors: Factors related to the personal attitudes and experiences of people during the application of RETSA and MRETS in the project might also have influenced the RE process model developed.

Additionally, it has to be emphasized that the purpose of the presented case study is not to formally validate the RESTA model, but rather to present the overall RESTA model and demonstrate its feasibility and its likely positive impact to the PSS software project.

We argue that based on the case study it is possible to state that:

- RETSA provided valuable help during the RE technique selection process.
- The RE techniques selected with the help of the RETSA model can be considered as well-suited for the PSS project.
- It can be assumed that the RETSA model will continue to be useful for the selection of RE techniques for other software projects within company X.

The developers in the organization stressed that the RE techniques selected by using the RETSA model and used in the software project contributed to the overall quality of the software requirements. And the high quality of the requirements made tracing requirements to their sources possible, understanding the requirements easier, and conflict resolution more effective compared to previous projects. Further detailed analysis of all the data in the case study is needed. And the further application of the RETSA model in the broader context of software project will be part of our future research.

4 Conclusion and Further Work

Research has shown that improving the software process by using suitable techniques can contribute to the overall quality of the software product [5, 31]. In this paper, we presented a RETSA model that helps select RE techniques based on the characteristics of the software project. The RETSA model is developed based on existing knowledge of RE techniques, investigation of literature, and the experience of software developers and experts from both industry and academia. An effort was made to take important factors into consideration, such as software project attributes and suitability of RE techniques with respect to the characteristics of software project. Some of the results of a case study were presented to show the help that the RETSA model provided to the PSS software project. The data collected from the PSS project supports the hypothesis made in the case study.

Our further work will refine the RETSA model so that it can accommodate more complicated software projects by considering more project attributes. Additionally, further research will be conducted to further improve the integration of the RETSA model within the MRETS methodology in the broader context of software projects. More case studies are still needed to analyze the effectiveness of the RESTA model. Finally, the development of a prototype that facilitates the use of the RETSA model and the MRETS methodology for the selection of RE techniques is a task for the near future.

References

[1] Kotonya G. and Sommerville I., RE, Processes and Techniques, John Wiley & Sons Ltd, 1998

[2] Damian, D.; and Chisan, J.; An Empirical Study of the Complex Relationships between Requirements Engineering Processes and Other Processes that Lead to Payoffs in Productivity, Quality, and Risk Management， IEEE Transactions on Software Engineering, Volume 32, Issue 7, July 2006, pp. 433 – 453

[3] Brooks F., "No Silver Bullet: Essence and Accidents of Software Engineering," Computer, Vol. 20, No. 4, Apr. 1987, pp. 10-19.

[4] Claus C., Freund M., Kaiser M., and Kneuper R., "Implementing Systematic Requirements Management in a Large Software Development Programme," Proceeding of Fifth Intternationl Workshop on Requirements Engineering, Foundation of Software Quality, 1999, pp. 33-42.

[5] El-Emam KE. and Birk A., "Validating the ISO/IEC 15504 Measure of Software Requirements Analysis Process Capability," IEEE Trans. Software Eng., vol. 26, no. 6, Nov./Dec. 2000, pp. 541-566.

[6] Extreme Chaos (2001), the Standish Group International, Inc.,

[7] Extreme Chaos (2003) the Standish Group International, Inc.,

[8] Hall T., Beecham S., and Raner A., "Requirements Problems in Twelve Software Companies: An Empirical Analysis," IEE Proc. Software, Vol. 149, No. 5, 2002.

[9] Dobson JE., Blyth AJC., Chudge J. and Strens R. (1994) The ORDIT Approach To Organizational Requirements In Jirotka, M. and Goguen, J. (Eds.), Requirements Engineering: Social and Technical Issues, Academic Press, pp. 87-106

[10] Jiang, L, A Framework For Requirements Engineering Process Development, University of Calgary, PhD Thesis, Sep. 2005

[11] Mannio M. and Nikula U., Requirements Elicitation Using a Combination of Prototypes and Scenarios, Technical Report. Telecom Business Research Center, Lappeenranta University of Technology, P.O.BOX 20, FIN-53851 Lappeenranta, Finland, Lappeenranta, 2001, ISBN 951-764-528-7

[12] Sutcliffe A., A technique combination approach to Requirements Engineering. 3rd IEEE International Symposium on Requirements Engineering (RE'97) , January 05 - 08, 1997, Annapolis, MD.

[13] Jiang L. Eberlein A. and Far BH, Combining Requirements Engineering Techniques –Theory and Case Study, ECBS 12th IEEE International Conference on the Engineering of Computer-Based Systems, Greenbelt, MD., USA, April 2005

[14] Jiang L. Eberlein A. and Far BH. A Methodology for RE Process Development, ECBS 11th IEEE Int'l Conf. & Workshop on the Engineering of Computer-Based Systems, Brno, Czech Republic, May, 2004

[15] Jiang L. and Eberlein A. (2003) Requirements Engineering: A Review and A Proposal, the Proceedings of the Third ASERC Workshop on Quantitative and Software Engineering, Feb. 17-18, 2003, Banff, Alberta, Canada

[16] Jiang L. Eberlein A., Clustering Requirements Engineering Techniques, IASTED International Conference on Software Engineering and Applications (SEA 2006), to be held from November 13 to November 15, 2006, Dallas, USA

[17] Jiang L. Eberlein A., Far BH., Majid M., An Innovative Requirement Engineering Technique Selection Model, Technical Reports, August, 30, 2004, Dept. of Electrical and Engineering, University of Calgary, http://www.enel.ucalgary.ca/~jiangl /research/indexOfResearch.htm

[18] Jones C. (2000) Software Assessment, Benchmarks, and Best Practices. Addison-Wesley

[19] Pfeiffer S. (2003) Requirements Engineering for Dynamic Markets, MSc. Thesis. Department of Electrical And Computer Engineering, University of Calgary, Calgary, Alberta

[20] McPhee C. (2000). Requirements Engineering for Projects with Critical Time-To-Market. M. Sc. thesis, University of Calgary.

[21] Bray IK. (2002) An Introduction to Requirements Engineering, Addison Wesley

[22] Haywood E. and Dart P. (1999) Analyzing Projects to Decide How To Model the Requirements, Proceedings of The Fourth Australian Conference on RE, Sep. 1999, pp149-159, CSIRO-Macquarie University JRCASE

[23] Yourdon E. (1997) Death March The Complete Software Developer's Guide to Surviving "Mission Impossible" Projects, Prentice Hall PTR

[24] Salzer H., ATRs (Atomic Requirements) Used Throughout Development Lifecycle, presented at the 12th International Software Quality Week 1999

[25] Boehm BW., Software Engineering Economics, Prentice-Hall, Englewood Cliffs, NJ 1981

[26] Mesiar R, Calvo T, Mayor G (2002), Aggregation Operators: New Trends and Applications, Physica-Verlag

[27] Jain AK, Dubes RC (1998) Algorithms for Clustering Data. Prentice-Hall

[28] Wohlin C., Runeson P., Ohlsson MC., Regnell B. and Wesslen A. (2000) Experimentation in Software Engineering: An Introduction, Kluwer Academic Publishers

[29] Kitchenham B., Pickard L., and Pfleeger SL. (1995) Case Studies for Method and Tool Evaluation, IEEE Software. pp. 52-62

[30] Glass RL. (1994) The Software Research Crisis,

[31] Broadman J. and Johnson D., "Return on Investment from Software Process Improvement as Measured by U.S. Industry," Crosstalk, vol. 9, no. 4, pp. 23-29, 1996.

Supporting UML Sequence Diagrams Using a Processor Net Model

Tony Spiteri Staines
Department of Computer Information Systems, University of Malta
toni_staines@yahoo.com

Abstract

This paper describes how UML sequence diagrams can be supported using an executable processor net. Distributed real time transaction processing systems require validation, verification and performance analysis. A method for doing this is presented creating a processor net for a flight reservation. Some advantages of this approach are the creation of executable models, schemas, formalization and performance analysis .Other issues are discussed.

1. Introduction

1.1. Notations and the UML

The UML notation is used for designing different applications ranging from embedded systems [3,10] to distributed information systems. Requirements engineering and performance engineering [27] use formal system verification [25], modeling, performance analysis, viewpoints and conceptual patterns. These are difficult to cover by a single method, where developers might opt for sub-optimal solutions. A MDE approach tries to tackle these problems [26]. Many software engineering methods lack the generation of executable models [23].

UML dynamic diagrams need to be tested for consistency [6]. In one instance an execution algorithm for UML activity diagrams based on the semantics of UML state machines dealing with real time behavior was used [7]. In [5,16,24] it was suggested how to derive performance models from UML diagrams. In [22] UML diagrams are converted into generalized semi markov processes.

1.2. Petri nets and the UML

There are different proposals how to formalize the UML using Petri nets [1,2]. Distributed manufacturing systems have been successfully modeled using the UML and Petri nets [4]. Petri nets and the UML have been combined for structural analysis, performance statistics and to generate process algebras. Many classes of Petri nets exist creating a selection problem. Petri nets used in [4,11] can be converted to generalized stochastic Petri nets [22]. Place transition nets are more appropriate for structural analysis. Colored Petri nets [15,18,19,20], object oriented Petri nets [1,2,17] and higher order nets [13,14] have many enhanced properties. These could be data tokens, parameters, input ports, output ports, procedures, functions, containers. They are designed for modeling complex activities and message sequencing.

UML diagrams for translation must be selected. Interaction diagrams or those representing communication between object classes and activity diagrams are suitable for translation.

1.3. B language and the UML

There are problems to translate UML into B language constructs addressed in [28]. There is no mechanism to 'lift' a specification in B and other specification languages, unlike Z which allows this to be done. Z schemas can be used to define behavior which can be promoted to a higher level. The processor net approach promotes the use of schemas. Schemas can be defined for processors using the Z language or VDM thus 'lifting' off a specification. Hence the problem when using B is tackled.

1.4. A processor net approach

The focus in this paper is to translate sequence diagrams into a processor-net. The processor-net model is a simplified version of the actor model presented in [12]. This model uses ideas from place transition nets and higher order nets. Tokens can have different values like integer, str, boolean, record or even objects. Transitions are called processors. Places are channels or stores. Channels can contain 'tokens'

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

from a specified data type. Basic types can be used to define more complex types. Processors can contain program code.

2. Application Scenario

Some local flight reservation systems were investigated. These connect to global distribution systems (GDS) accessible via different portals. Flight reservation involves complex rules and different costs. There are complex i) functional and ii) non-functional requirements. E.g. the maximum time to create a transaction should never exceed three minutes. Every three minutes. Certain airline companies send 'availability updates' to a central server for flight reservation. There are operational issues e.g.: i) Once a Seat-Sell is submitted there is immediate booking. ii) Delete or modification update requires immediate notification.

The most important operations are related to the flight booking process. These are i) View flight schedules, ii) Create flight reservation, iii) View flight reservation details iv) Modify/cancel flight reservation details. The reservation process involves many detailed steps and verification.

The main GDS providers are: Amadeus, Sabre, Galileo and Worldspan. These systems operate on legacy platforms. There are new trends where companies like ITA software are offering optimized and flexible solutions that are service oriented and configurable.

Flight reservation can be classified into i) a traditional approach. Here the steps are carried out sequentially: *search request, check seat availability, check price and reserve flight*, ii) a service oriented approach where *search, seat check, price check could be combined.* Fig. 1 depicts the normal approach and fig. 2 depicts the service oriented approach combining only searching and seat checking in one step.

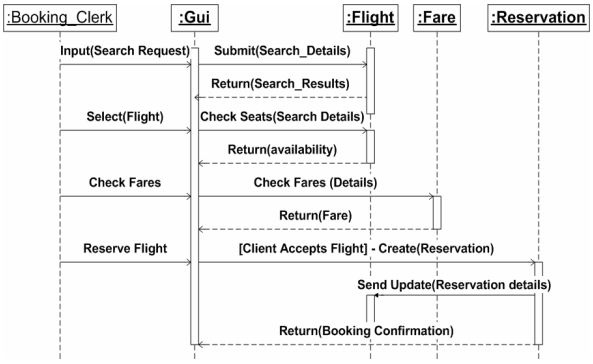


Figure 1. Normal flight reservation

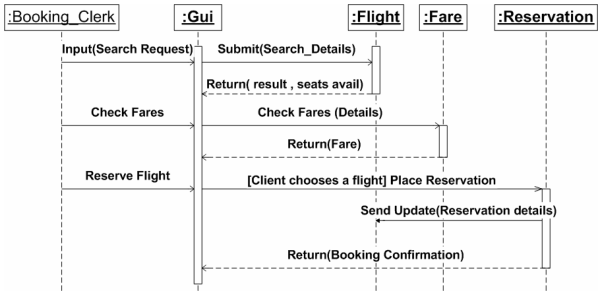


Figure 2. Modified flight reservation

Fig. 1 is used for obtaining the processor net. This is because it has more detail. Several combinations are possible from the steps used for flight reservation. These depend on the global distribution system used.

3. Building and executing the processor net

3.1. Sequence diagram conversion

Fig. 1 describes part of the normal booking and reservation process. These are performed within a defined period of time and some form of acknowledgement has to be received by the system.

The steps/ algorithm in fig. 3 convert the properly defined UML 2 sequence diagram (fig. 1) into an empty processor net diagram (fig. 4). A conventional approach to modeling sequence diagrams using Petri nets is to model message communication as in [11]. It is also possible to treat classes as processors. A different main actor based concept is used here.

Preconditions : A properly labeled UML 2 Sequence Diagram
Input: Sequence Diagram, identified class
Output: Processor – Net Diagram (Elementary Flat Processor Net Definition)

START

FOR EACH message m∈ M sent by the identified class to the right hand side exclusively
DO
Insert new processor p
Insert new channel c
Insert input flow/arc from c→ p
OD

FOR EACH return message m∈ M received by the identified class from the right hand side classes exclusively
DO
Insert new channel c
Identify proper processor $p_n \in P$ where {n = 1.. total no of processors}
Insert output flow/arc from p → c
OD

FOR EACH processor $p_x \in P$ where {x = 1.. total no of processors -1}
DO
Insert new channel c
Insert output flow/arc $p_x \rightarrow c$
Insert input flow/arc $c \rightarrow p_{x+1}$
OD

STOP

Figure 3. Sequence diagram conversion algorithm

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

The 'actor' initiates the whole process of message communication with the other classes via another main class like the **:GUI**. In this example the **:GUI** class is the identified class for the algorithm in fig. 3.

The algorithm's function is to identify all important messages sent and received by the identified actor class and construct a processor net. With a slight modification it is possible to generate also a place transition net.

The result of the algorithm in fig. 3 is defined as a flat net three tuple (P,F,C) model that is not decomposable. Where P = finite set of processors, P = {p1,p2….p_n}, $P \neq \phi$, F= Set of directed flows/arcs from a processor p to a channel c or from a channel c to a processor p; $F \subseteq \{(CxP) \cup (PxC)\}$; $F \neq \phi$;C= finite set of channels ,C ={c1,c2,…c_n} , $C \neq \phi$, F,P,C are mutually disjoint.

The model obtained is still empty. Some additional development is required to construct a fully executable processor net. In [12] rules for tokens, firing, data types and complex class types are explained. Processors can behave deterministically or non-deterministically.

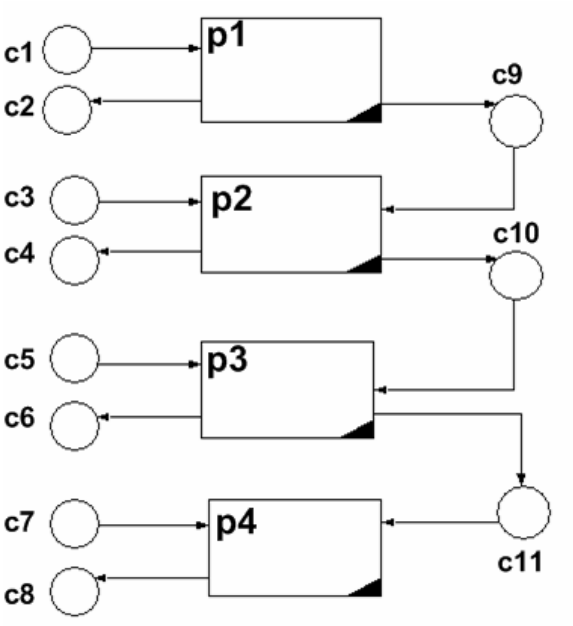


Figure 4. Initial processor net

3.2. Constructing an executable model

The completed processor net Diagram is built and coded using an executable specification tool called EXSPECT [8,9]. The following sequence of steps describes how this is done:

Step1: Construct the empty processor net.
The empty processor net in fig. 4 is built using channels, processors and connectors.

Step2: Rename channels and processors.
This step is optional. The channels and processors are correctly named to represent the real system, making it more readable.

Step2: Add stores if required.
Stores are similar to channels but they can have data types representing different system states and data similar to databases. Three stores are easily identified from the sequence diagram, these are: Flight_Store, Fare_Store and the Reservation_Store and have data types Flight,Fare and Reservation respectively.

Step3: Define and construct the data types to use.
Data types have to be properly identified. Different type constructors, sets, lists, mapping types and tuple types are possible. For this system types identified from the UML diagrams were used. A PNR (Passenger Name Record) type was created having details like hename, flight_code, seats, hecontact, hetk, herf, heeot. Other data types like flight ,fare, confirmation were also created, these are bound to the tokens.

Step4: Code the processors.
Processor specification uses a functional specification language [8].Special functions and constructs based on sets can be specified. The following tasks are performed to code the processor: i) Define inputs, ii) Define outputs, iii) Define a value expression for the processor. The input and outputs are the channels & stores. If step 3 is performed this is not necessary. Only the processor main definition is required. The definition processes the input tokens. The processor consists of a specification that builds the output value from the input value bound to the tokens. e.g. *out* <— *in* would copy the value of *in* channel to the *out* channel. The code in section 4 is a comprehensive example.

Step5: Executing the model
The final model shown in fig. 5 is executed as follows: i) Place tokens with appropriate values in the channels & stores, ii) fire the processor, iii) check the output produced in the output channels & stores. A processor is activated only when its preconditions are met. This implies that its input channels & stores have data that satisfies the processor definition for activation. On activation the input tokens are consumed and output tokens are created and placed accordingly in the output channels.

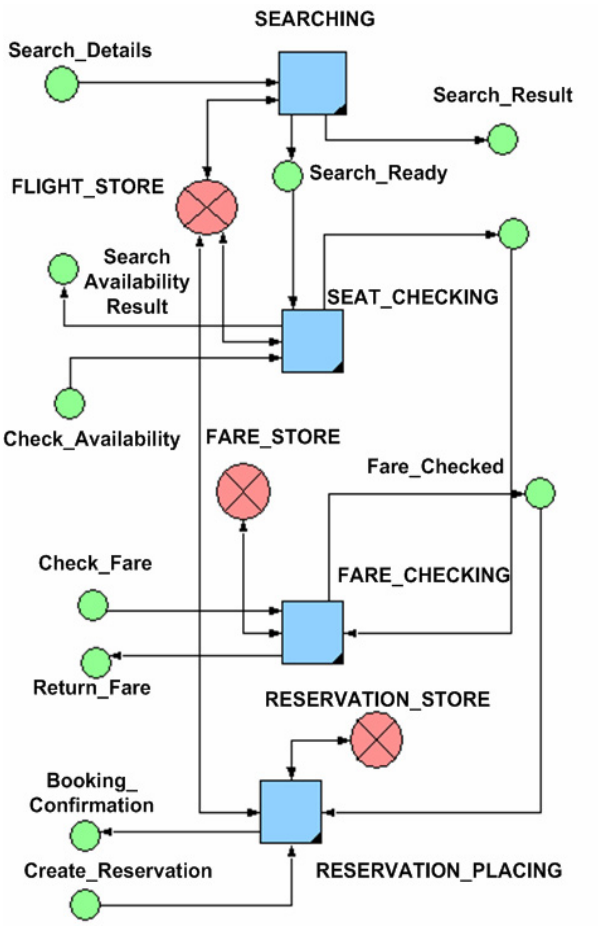


Figure 5. Final processor net

4. Deriving formal specification schemas

The processor specification is in a format that is suitable for deriving formal specification schemas having useful mathematical properties [12,21]. Schemas can be generated for data types, channels, stores, restrictions, etc. A similar approach can be carried out for each processor in the system. More complex functions and detailed schemas for each part of the system e.g. the channels and stores, can be created. The schemas could also be developed using Z or VDM.

4.1. Processor reservation placing

The code below is a simplified version of processor: RESERVATION_PLACING.

result= (pick(set([x:Flight_Store|x@flight_code= Create_Reservation@flight_code]))
// checks the reservation request with flight details

seat_check= result@seats>= Create_Reservation@seats
// checks for flight match with the reservation entered and that seats are available .

update_flight_store= Flight_Store< [flight_code:result @flight_code),avail:result@avail),seats:result@seats) - Create_Reservation@seats]ins(set([x:Flight_Store|x@ flight_code!=Create_Reservation@flight_code]))
// update the flight_store contents with the reservation details

--Processor main implementation--

if seat_check= true then

Reservation_Store<—Create_Reservation ins Reservation_Store,
// add the new reservation to reservation store

Booking_Confirmation<— Create_Reservation,
// send booking confirmation result

update_flight_store

fi

The implementation is converted into a schema. Primitive functions defined are: i) a Projection function Π that selects values from rows and tuples (e.g. Π_2 (a,b,c,d) yields b , ii) a Set function that returns a set of members of x for a specific condition Set[x:T->bool]:$x, iii) a Pick function that converts a set into a tuple Pick[x:$T]:T [8,9,12]. Below is part of such a schema for the reservation placing. This schema could be verified.

4.2. Processor schema

Processor : RESERVATION_PLACING

a? : Create_Reservation
b! : Booking_Confirmation
c' : Flight_Store
d' : Reservation_Store
x : Pick(Set(c' | Π_{Flight_code}(c')= Π_{Flight_code}(a?)))

if Π_{seats}(x) >Π_{seats}(a?) then
d' = {a?} $\cup$ d'
b! = a?
c' = {[$\Pi_{Flight\_code,\ avail}$ (x), Π_{seats}(x)- Π_{seats}(b?)]} $\cup$ {c'/[$\Pi_{Flight\_code,avail,seats}$ (x)]}
fi

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

5. Performance measures and optimizing

The processor net must be transformed before alternative combinations can be evaluated.

5.1. Processor net transformation method

The processor net has acyclic behavior. It is possible to transform the processor net for time analysis.

The processor net developed is composed of a set of ordered processes that execute in a particular order. The processor net is modified as follows: i) all channels are given a similar data type, ii) channels that do not connect the processors are removed. iii) processors are modified to contain time, iv) A firing cycle is introduced if required, v) a random store is added giving tokens random time values from a range [e_i^-, e_i^+]. Where the values e_i^- and e_i^+ are the min, max processing time. The idea uses concepts from Petri net theory [29], ensuring there are place and transition invariants and feedback mechanisms [31,32]. It is possible to transform the processor net into a Time place transition net.

5.2. Rules for alternative combinations

Processor net processors are ordered sequentially. For finding alternative combinations the following rules apply: i) Only two types of processor combination are possible: sequential or parallel. ii) The processor net is always choice free and conflict free. iii) All processors are functional. iv) The processor net has at least one directed path from the initial processor to the terminating processor. v) All processors must be in the directed path. vi) Precedence constraints for the processors are observed. Combining processors cannot result an increase in the number of steps in the system.

A large sized net can be reduced or simplified by combining processors and channels using Petri net reduction rules that preserve liveness and boundedness [12,32]. These rules restrict the number of possible combinations. Combining processors also depends on the business rules. These two steps indicate how to find combinations i) Define the transformed processor net. ii) Find alternative configurations using the given rules. If two processors a,b execute in parallel and $time(P_a) < time(P_b)$ then the critical path or cycle time is determined by processor b. So for time analysis processor 'a' could be omitted [12].

5.3. Evaluating different configurations

Analysis techniques for evaluating the configurations are classified into i) simple and ii) advanced.

Simple techniques focus on one particular measure like time. The transformed processor net can be represented as an acyclical directed bipartite graph or activity network. Time analysis and critical path analysis can be performed. It is possible to add a cycle to the net and use cycle time analysis as is done for Time Petri nets. Simulation models for different configurations can be built and results compared. Another simple method is to use a cost function for the processors in conjunction with predecessor and timing constraints and try to minimize the operational cost. The total cost function to be minimized can be represented as $\sum_{p=1}^{n} C_p T_p \leq buget$, $p = 1..n$, where p is the processor number and n the maximum number of processors , C_p is the unit time cost of processor p , T_p is the average processing time for processor p.

Advanced complex optimization could be formulated as a combinatorial optimization problem (COP). Some COP can be solved using integer programming or heuristic algorithms [30].

6. Results

6.1. Modeling the behavioral sequence

If the techniques proposed in [11] are used to convert the sequence diagram in fig. 1, the result is that of a large Petri net having many transitions and places. Classes can have messages sent that require certain control and actions because of nested sequence of events. The processor net approach simplifies this. In given example there are only four main processors. Processors can contain detailed programming logic that is useful for testing an executable model.

Appropriate data was placed in the stores, data tokens that represent actions were added and processors were enabled and fired getting results. Processor RESERVATION_ PLACING is enabled by having data tokens in Flight_Store, Create_Reservation and Fare_Checked. These are shown in the 'before' column in Table 1. After firing, it updates the details in the Flight_Store by reducing the number of seats available by the number that was booked. A new reservation message added to the Reservation_Store and a reply message is placed in the Booking_Confirmation Channel. Similar results were obtained for all the processors.

Table 1. Processor reservation placing

STORES	BEFORE	AFTER
Flight_Store	{[avail:'YES',flight_code:'KLM105',seats:150]}	{[avail:'YES',flight_code:'KLM105',seats:148]}
Reservation_Store	{}	{[hename:'NM1 SMITH J',flight_code:'KLM105',seats:2,hecontact:'AP92949693',hetk:'TKTL01MAR' herf:'RF JOE',heot:'ER']}
CHANNELS		
Create_Reservation	hename:'NM1 SMITH J',flight_code:'KLM105',seats:2,hecontact:'AP92949693',hetk:'TKTL01MAR' herf:'RF JOE',heot:'ER'	
Fare_Checked	1	
Booking_Confirmation		hename:'NM1 SMITH J',flight_code:'KLM105',seats:2,hecontact:'AP92949693',hetk:'TKTL01MAR' herf:'RF JOE',heot:'ER'

6.2. Performance analysis

Fig. 6 shows two different configurations. In configuration A two processors SEAT_CHECKING and SEARCHING have been placed to operate in parallel. Searching & seat_checking can be reduced to a single processor yielding identical results, refer to section 4.1. This is not possible with every GDS, but some modern ones allow different configurations and operational logic. E.g. QTX developed by ITA software.

Table 2. Processor max and min time

PROCESSOR	Min Time (s)	Max Time(s)
SEARCHING	2	9
SEAT_CHECKING	2.9	10
FARE_CHECKING	2	8
RESERVATION_PLACING	20	40
REPEAT	0	0

Table 3. Cumulative reservation time

No. of System Cycles	Time config. A	Time config. B
10	373	418
30	1179	1308
70	2848	3166
100	4099	4565
150	6289	6994
200	8307	9251
250	10447	11610
300	12579	13977

Configuration A is an improvement over configuration B. The average cycle time is 42 seconds for configuration A and 46 seconds for configuration B. This is obtained from table 3.

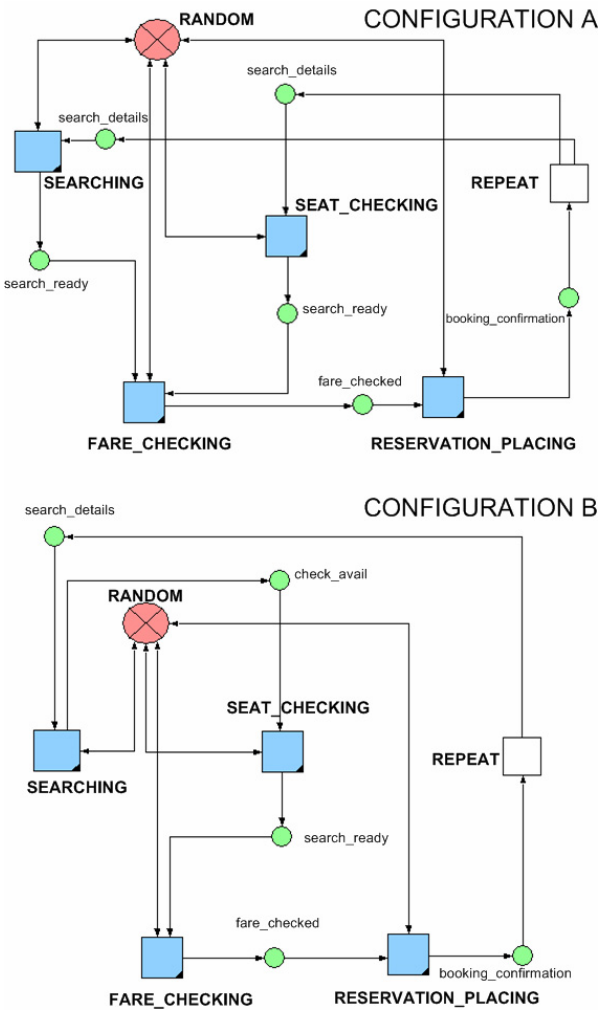


Figure 6. Two alternative configurations

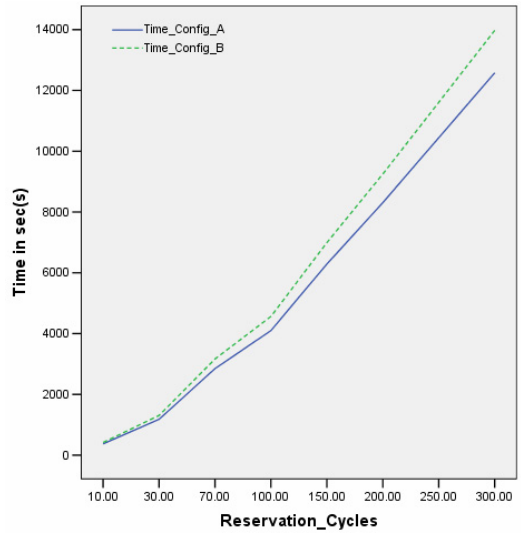


Figure 7. Cumulative reservation time

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

The data was obtained from 300 system cycles, using the timings from table 2. Fig. 6 depicts the results of table 3 graphically. A random distribution for time values was used. Table 2 shows the max and min times for each processor. The models in fig. 6 can be analyzed using methods applicable to Petri nets.

7. Conclusion

The idea presented is to find a practical way to translate a UML Sequence Diagram into a 'Processor-Net' model. It is possible to create i) executable specifications, ii) define schemas, iii) generate performance measures for alternative configurations. The processor net model is more compact than a Petri net. It can be translated into a Petri net or a Colored Petri net. Specifications can be defined for each processor. These are based on formal definitions and proofs. The processor net can be developed to create a detailed 'Actor Model' [12].

There are some limitations i) the algorithm described will not work for all UML sequence diagrams, ii) complex nets could become difficult to optimize. Also for flight reservation there are many other factors involved. E.g. Price to seating relationships can be very complex and time dependant.

This approach can be useful for describing special classes of real time systems e.g. manufacturing systems, network based systems, distributed information systems, automated systems and mobile computing. However more research is indicated. .

8. References

[1] L. Baresi, M. Pezze, "On Formalizing UML with High-Level-Petri Nets", *Concurrent object-oriented programming and petri nets: advances in Petri nets*, Springer-Verlag, ISBN 3-540-4192-x, 2001, pp. 276-304.

[2] L. Baresi, "Some Hints on Formalizing UML with Object Petri Nets", *6th World Conferenece Proceedings on Software Engineering and Knowledge Engineering*, Ischia Italy, 2002.

[3] G. Booch, *Object-Oriented Design with Applications*, 2d ed. Reading, Mass.: Addison-Wesley, 1991.

[4] B.Bordbar, L. Giacomini, D.J. Holding, "Design of Distributed Manufacturing Systems using UML and Petri Nets",*6th International Conference on Intelligent Systems*, Federation of Automatic Control(IFAC) Spain, May 2006, pp. 91-96.

[5] C.Canevet, S.Gilmore, J. Hilliston, L Kloul, P. Stevens, "Analysing UML 2.0 Activity Diagrams in the Software Performance Engineering Process", *DEGAS Project*, Lab. for Foundations of Computer Science Univ. of Edinburgh Scotland, Sep 2002.

[6] G. Engels, J. H. Hausmann, R. Heckel, S. Sauer, "Testing The Consistency of Dynamic UML Diagrams", *Proceedings of the 6th International Congress of Integrated Design and Process Technology* , IDPT, Pasadena CA USA, Jun 2002.

[7] R. Eshuis , R. Wieringa, An Execution Algorithm for UML Activity Graphs, *Lecture Notes in Computer Science*, Vol 2185, Springer-Verlag, 2001, pp. 47-61.

[8] Exspect User Manual, Technische Universiteit, Eindhoven, 1999.

[9] Exspect Tool, Technische Universiteit, Eindhoven.

[10] G. Frick, B. Scherrer, K.D. Muller-Glaser, "Designing the Software Architecture of an Embedded System with UML 2.0",*Proceedings of UML 2004 Workshop On Software Architecture Description & UML*, Portugal, Oct 11-15 2004.

[11] T. Gehrke, U.Goltz, H. Wehrheim, "The Dynamic Models of UML: Towards a Semantics and its Application in the Development Process", *Technical Report Informatik-Bericht 11/98*, University of Hildesheim, Germany, Sep 1998.

[12] K.M. van Hee, *Information Systems Engineering A Formal Approach*, University Press Cambridge, 1994.

[13] K. Hoffmann, T. Mossakowski, "Algebraic Higher-Order Nets: Graphs and Petri Nets as Tokens",*16th International Workshop WADT*, Lecture Notes in Computer Science, Vol. 2755, Springer – Verlag, , Nov 2003, pp. 253-267.

[14] J.W. Janneck, R. Esser, "Higher-Order Petri Net Modelling – Techniques and Applications", ACM International Conference Proceedings Series; Vol. 145 Proccedings of the conference on application & theory of Petri Nets: Formal Methods in Software Engineering and defense systems – Vol 12. (WSEFM), Adelaide Australia, Jun 2002, pp. 17-25.

[15] J.B. Jørgensen, "Coloured Petri Nets in UML-Based Software Development – Designing Middleware for Pervasive Healthcare" ,*Proc. 4th Int. Workshop on the Practical use of CPN & CPN tools,* Diami Technical Report PB-560, Aarhus Denmark, Aug 2002, pp. 61-80.

[16] C. Kabajunga, R. Pooley, "Simulating UML Sequence Diagrams", *Proc. of 15th UK Perf. Eng. Workshops*, Dept. of Computer Science Univ. of Bristol, Jul 1999,pp. 23-33.

[17] B. Krena, T. Vojnar, "Type Analysis in Object-Oriented Petri Nets", *Proceedings of ISM'01 Hradec nad moranici Czech Republic* (Marq Ostrava), 2001, pp. 173-180.

[18] L.M. Kristensen, S. Christensen, K. Jensen, "The Practioner's Guide to Coloured Petri Nets", *International Journal On Software Tools for Tech. Transfer (STTT)*, Vol. 2, Springer-Verlag,1998, pp. 98-132.

[19] L.M. Kristensen, K. Jensen, "Specification and Validation of an Edge Router Discovery Protocol for Mobile AD Hoc Networks", *Lecture Notes in Computer Science*, Vol. 3147,Springer – Verlag,2004, pp. 248 -269.

[20] L.M. Kristensen, J.B. Jorgensen, K. Jensen, "Application of Coloured Petri Nets in System Development", *Lecture Notes in Computer Science*, Vol. 3098 , Springer-Verlag, 2004, pp. 626-685.

[21] D. Lightfoot, *Formal Specification using Z*, Palgrave , NY, 2001, ISBN 0-333-76327-0, Ch 6 pp. 37 – 97.

[22] C. Lindemann, A. Thummler, A. Klemm, M. Lohmann, O.P. Waldhorst, "Performance Analysis of Time-Enhanced UML Diagrams Based on Stochastic Processes", *3rd ACM Workshop on Software & Performance Rome* , Jul 2002, pp. 25-34.

[23] J.W.S. Liu, *Real-Time Systems*, Pretence Hall, NJ, 2000.

[24] R. Pooley, "Using UML to Derive Stochastic Process Algebra Models", *Proc. of 14thUK Perf. Eng. Workshops,* Univ. of Edinburgh Scotland, Jul 1998.

[25] J.A. Saldhana, S.M. Shatz, Z. Hu, "Formalization of Object Behavior and Interaction From UML Models",International Journal of Software & Knowledge Engineering, 11(6), 2001, pp. 643-673.

[26] D.C. Schmidt, "Model-Driven Engineering*", IEEE Computer Model-Driven Engineering\\Smart Cameras*, Vol 39 No 2, Feb 2006, pp. 25-31.

[27] C.U. Smith, *Performance Engineering of Software Systems*, Addison-Wesley, 1990.

[28] C.Snook, M.Butler, "UML-B: Formal Modeling and Design Aided by UML", *ACM Transactions on Software Engineering and Methodology*, Vol 15 No 1, Jan 2006, pp. 92–122.
.

[29] G. Stemersch, R.K. Boel, "Structuring acyclical Petri Nets for Reachability Analysis and Control", *International Journal of Intelligent Control and Systems*, Vol 10 No 2, Jun 2005, pp. 175-187.

[30] L.A. Wolsey, *Integer Programming*, Wiley, NJ USA, 1998.

[31] K. Yamalidou, J Moody, M. Lemmon, P. Antsaklis, "Feedback Control of Petri Nets Based on Place Invariants", *Technical Report of the ISIS Group University of Notre Dame IN 46556*, ISIS-94-002, 1994.

[32] M. Zhou, K. Venkatesh, *Modeling, Simulation, and Control of Flexible Manufacturing Systems A Petri Net Approach*, Series in Intelligent Control and Intelligent Automation vol. 6 World Scientific, MA USA, 1999.

Reconciling Synthesis and Decomposition: A Composite Approach to Capability Identification

Ramya Ravichandar and James D. Arthur
Department of Computer Science
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24060
{*ramyar, arthur*}*@vt.edu*

Robert P. Broadwater
Electrical and Computer Engineering
Virginia Polytechnic Institute and State University
Blacksburg, Virginia 24060
dew@vt.edu

***Abstract*— Stakeholders' expectations and technology constantly evolve during the lengthy development cycles of a large-scale computer based system. Consequently, the traditional approach of baselining requirements results in an unsatisfactory system because it is ill-equipped to accommodate such change. In contrast, systems constructed on the basis of Capabilities are more change-tolerant; Capabilities are functional abstractions that are neither as amorphous as user needs nor as rigid as system requirements. Alternatively, Capabilities are aggregates that capture desired functionality from the users' needs, and are designed to exhibit desirable software engineering characteristics of high cohesion, low coupling and optimum abstraction levels. To formulate these functional abstractions we develop and investigate two algorithms for Capability identification: Synthesis and Decomposition. The synthesis algorithm aggregates detailed rudimentary elements of the system to form Capabilities. In contrast, the decomposition algorithm determines Capabilities by recursively partitioning the overall mission of the system into more detailed entities. Empirical analysis on a small computer based library system reveals that neither approach is sufficient by itself. However, a composite algorithm based on a complementary approach reconciling the two polar perspectives results in a more feasible set of Capabilities. In particular, the composite algorithm formulates Capabilities using the cohesion and coupling measures as defined by the decomposition algorithm and the abstraction level as determined by the synthesis algorithm.**

I. Introduction

The property of change-tolerance is of paramount importance in complex emergent systems. These computer based systems are of large magnitude, have lengthy development cycles and are envisioned to be utilized for an extended lifetime. In addition, their inherent complexity results in emergent behavior [1] that is often unexpected. For example, the introduction of a new functionality in the system may result in unanticipated interactions with other existing components that can be detrimental to the overall system functionality. Moreover, in order to function satisfactorily complex emergent systems must accommodate the effect of dynamic factors such as varying expectations of the stakeholders, changing user needs, technology advancements, scheduling constraints and market demands, during their lengthy development periods. We conjecture that these changes can be accommodated with minimum impact, if systems are architected using aggregates that are embedded with change-tolerant characteristics. We term such aggregates as *Capabilities*. Capabilities are functional abstractions that exhibit high cohesion, low coupling and balanced abstraction levels. The property of high cohesion helps localize the impact of change to within a Capability. Also, the ripple effect of change is less likely to propagate beyond the affected Capability because of its reduced coupling with neighboring Capabilities. An optimum level of abstraction assists in the understanding of the functionality in terms of its most relevant details [2]. In addition, we observe that the abstraction level is related to the size of a Capability; the higher the abstraction level, the greater is the size of a Capability [3]. From a software engineering perspective, abstractions with a smaller size are more desirable for implementation. Therefore, we need to design an algorithm based on the three characteristics of cohesion, coupling and abstraction, that in some sense, "optimizes" the identification of Capabilities. Specifically, we use a top-down and a bottom-up approach as the basis of the algorithms for formulating Capabilities. This is because our cognitive ability to examine a problem from both a top-down and a bottom-up perspective facilitates the application of widely diverse solution approaches. This phenomenon is evident in the field of software engineering where development strategies such as top-down design, bottom-up testing, top-down integration and others that incorporate a top-down or a bottom-up perspective are utilized in the different stages of system development. In particular, for Capability identification we focus on needs analysis, a phase prior to requirements specification, because Capabilities are formulated from user needs. At this point we consider only the functional aspects of the system. Following convention, we develop two algorithms for Capability identification that are based on the top-down and bottom-up approaches:

- *Synthesis:* This is an algorithm based on the bottom-up approach. The system is understood in terms of its most detailed elements, which are then systematically aggregated to form abstractions of higher levels.
- *Decomposition:* This is an algorithm based on the top-down approach. The system is visualized in terms of its highest level mission, which is then systematically decomposed into abstractions that are more detailed.

In either approach the objective is to identify functional abstractions that are maximally cohesive and minimally coupled

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

as Capabilities. We assessed the efficacy of the synthesis and decomposition algorithms by executing them on a real-world computer based library system. Our empirical analysis reveals that neither approach is sufficient by itself to determine the best set of Capabilities. More specifically, the cohesion measure based on the synthesis approach is inordinately subjective. Additionally, the synthesis strategy provides little information to assist coupling measurements. However, this approach identifies aggregates of reduced sizes as Capabilities. In contrast, the decomposition approach expedites the measurement of cohesion and coupling but results in Capabilities that are of increased sizes. In other words, these Capabilities are defined at very high levels of abstraction. Therefore, we construct a composite algorithm to establish an equilibrium between the two polar approaches. This algorithm is based on a complementary approach that incorporates elements of cohesion and coupling from the decomposition strategy, and models abstraction from the synthesis perspective.

The remainder of the paper is organized as follows: Section II discusses related work and outlines the overall process of engineering Capabilities. In Section III and Section IV, we compare and contrast the three primary elements that determine a Capability — cohesion, coupling, and abstraction level — from the synthesis and decomposition approaches, respectively. In Section V, we describe our composite algorithm that combines the two approaches. Our conclusions are presented in Section VI.

II. Background

A system operating in the real world is subject to dynamic factors of change. These factors necessitate system evolution, the process of constantly adapting to various influences in order to function satisfactorily [4]. Software development processes that are ill-equipped to accommodate change are primarily afflicted with requirements volatility [5]. This phenomenon is known to increase the defect density and affect project performance resulting in schedule and cost overruns [6] [7]. Traditional Requirements Engineering (RE) strives to manage volatility by baselining requirements. However, the dynamics of user needs and technology advancements during the extended development periods of complex emergent systems discourage fixed requirements. More recently, techniques such as the Performance based specifications [8] [9] and Capability Based Acquisition (CBA) [10] are being utilized to mitigate change in large-scale systems. Performance based specifications are requirements describing the outcome expected of a system from a high-level perspective. The less-detailed nature of these specifications provides latitude for incorporating appropriate design techniques and new technologies. Similarly, CBA is expected to accommodate change and produce systems with relevant capability and current technology. It does so by delaying requirement specifications in the software development cycle, and by allowing time for a promising technology to mature so that it can be integrated into the software system. However, the Performance based specification and the CBA approaches lack a scientific procedure for deriving system specifications from an initial set of user needs. Moreover, they neglect to define the level of abstraction at which a specification or Capability is to be described. Thus, these approaches propose solutions that are neither definitive, comprehensive nor mature enough to accommodate change and benefit the development process for complex emergent systems.

Our approach, the Capabilities Engineering (CE) process, architects change-tolerant systems on the basis of optimal sets of Capabilities. In fact, Rowe and Leany suggest that it is beneficial to address the issues of evolution when modeling the system architecture [11]. Therefore, we design Capabilities to incorporate evolutionary-friendly characteristics such as high cohesion, minimal coupling, and pragmatic levels of functional abstraction. Figure 1 illustrates the two major phases of the CE process.

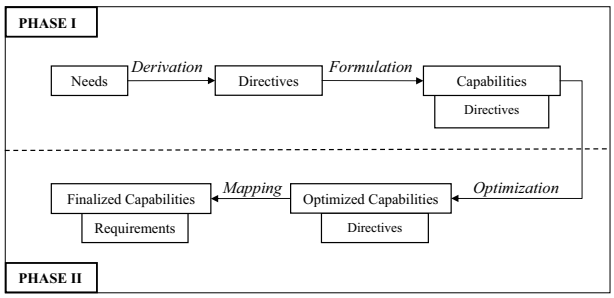


Fig. 1. *Capabilities Engineering Process*

Phase I identifies sets of Capabilities based on the values of cohesion, coupling and abstraction levels. Techniques of modularization suggest that high cohesion and low coupling are typical of stable units [12] [13]. Stability implies resistance to change; in the context of CE, we interpret stability as a property that accommodates change with minimum ripple effect. Ripple effect is the phenomenon of propagation of change from the affected source to its dependent constituents. Specifically, dependency links between aggregates behave as change propagation paths. The higher the number of links, the greater is the likelihood of ripple effect. Because coupling is a measure of interdependence between units [14] we choose coupling as one indicator of stability of an aggregate. In contrast, cohesion — the other characteristic of a stable structure — depicts the "togetherness" of elements within an aggregate. A unit is said to be highly cohesive if each of its elements is directed towards achieving a single objective. As a general observation as the cohesion of a unit increases, the coupling between the units decreases. However, this correlation is only approximate, and thereby, cannot be used to estimate the values of cohesion and coupling [13]. Therefore, we develop specific metrics to compute these values for potential Capabilities.

Phase II, a part of our ongoing research, further optimizes these initial sets of Capabilities to accommodate schedule constraints and technology advancements. In this paper, we focus on identifying Capabilities as outlined by Phase I.

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

In the following sections, we discuss the synthesis and the decomposition algorithms for computing Capabilities. We then explain the necessity for a composite algorithm that includes elements of cohesion, coupling, and abstraction from both these approaches.

III. Synthesis

The objective of the synthesis algorithm is to formulate Capabilities — functional abstractions with high cohesion and low coupling — from user needs that are obtained during the process of elicitation [15]. Needs are affiliated with the problem domain and requirements are associated with the solution domain. Capabilities are computed after the analysis of user needs but prior to requirements specification. We envision that by doing so Capabilities can bridge the chasm between the problem and the solution space, also described as the complexity gap [16]. It is recognized that this gap is responsible for information loss, misconstrued needs, and other detrimental effects that plague system development [17] [18].

The synthesis algorithm is based on a bottom-up approach, and hence, envisions a system in terms of its details. In particular, we consider system details that are defined at low levels of abstraction and are stated from a user's perspective. We term these details as *directives*. More specifically, a directive is a system specification that is described using the terminology of the problem domain. In contrast, a requirement is a system specification stated in the technical language of the solution domain. However, both a directive and a requirement share the commonality of being defined at a low level of abstraction.

Directives are a natural derivative of user needs. We use the directives as input to the synthesis algorithm for formulating Capabilities because they serve three main purposes. Firstly, directives strive to alleviate loss of domain knowledge, which has been identified as an important problem in RE [17]. They do so by describing system functionality in terms of the problem domain. This assists in capturing domain information. Secondly, directives are utilized to compute the cohesion and coupling values of potential Capabilities. Recall that optimal sets of Capabilities are to be determined from different functional abstractions. Capabilities are essentially system functionalities, and hence, are associated with one or more directives. Therefore, the cohesion and coupling measures of Capabilities are determined using directives. Lastly, directives facilitate the mapping to system requirements. Note that Capabilities only provide a high-level architecture based on system functionalities, and therefore, requirement specifications are still necessary to direct system development. Thus, directives are easily mapped to requirements because both entities are defined at similar levels of abstraction.

A. Algorithm

The synthesis algorithm aims to identify abstractions with maximum cohesion and minimum coupling, as Capabilities. In particular, it strives to maximize functional cohesion, the most desirable cohesion among all other types of cohesion (coincidental, logical, temporal, procedural, communicational, and sequential) [19]. This objective of the synthesis algorithm is illustrated in Figure 2. If every element of a unit is essential

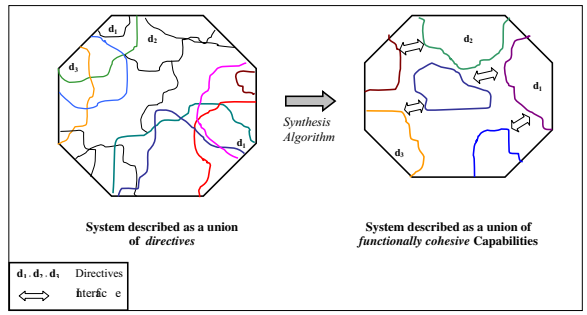


Fig. 2. *Objective of Synthesis Algorithm*

to the performance of a single function, then that unit is said to exhibit high functional cohesion [13]. Therefore, the first step of the algorithm enumerates functions that possess high functional cohesion. More specifically, we examine the significance of each directive in accomplishing various system functions. We use these significance values to compute the cohesion of a function in terms of all its participating directives. However, it is possible that the same function is described at different levels of abstraction. We represent the functions using Venn diagrams to visually understand and resolve the discrepancies in the abstraction levels. The algorithm is explained in detail next.

Let $d_1, d_2, \ldots, d_n$, $n \in \mathbb{N}$, denoting directives derived from user needs be the input to the synthesis algorithm. For each d_i perform the following steps to determine the Capabilities of a system:

1) *Identify all possible functions to which directive d_i contributes.* The relevance of a directive in accomplishing a function is estimated using the impact categories shown in Table I. This classification is intended to assess the impact of risks on a project [20]. The failure to implement a directive is also a risk, and thereby, we use this classification to determine the significance of a directive in implementing a system functionality. We assign relevance values based on the perceived significance of each impact category; these values are normalized to the [0,1] scale.

TABLE I
Relevance Values

Impact	Description	Relevance
Catastrophic	Task failure	1.00
Critical	Task success questionable	0.70
Marginal	Reduction in performance	0.30
Negligible	Non-operational impact	0.10

Formally, we enumerate the list of functions f_{im}, $m < n$, that d_i is associated with, as $InitialSet_i = \{f_{i1}, f_{i2}, \ldots, f_{im}\}$. For example, let d_1 help achieve functions

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

$f_{1j}, j = 1, \ldots, 7$. A Venn diagram representation indicating the different abstraction levels of the functions of $InitialSet_1$ is shown in Figure 3. Late, we use the relevance values of directives later to compute the cohesion of potential Capabilities.

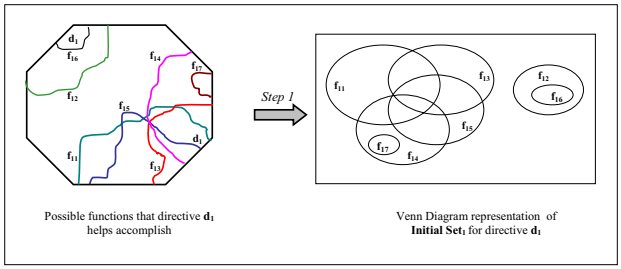


Fig. 3. *Example Initial Set for directive* d_1

2) *Expected system functionalities deduced from user needs can be stated at different levels of abstraction.* Consequently, certain functions constituting $InitialSet_i$ may be inclusive of other functions in the same $InitialSet_i$. For example, in Figure 3, f_{12} is inclusive of f_{16}. We avoid considering functional abstractions that are partially or completely redundant as potential Capabilities by constructing $Subset_i \subseteq InitialSet_i$ where $Subset_i = \{f_{ix} | f_{ix} \supseteq f_{iy}, \forall f_{iy} \in InitialSet_i; 1 \leqslant x, y \leqslant m\}$. Note that the functions in $Subset_i$ are not encompassed by any other function in $InitialSet_i$. This implies that $Subset_i$ consists of functions defined at the *highest* level of abstraction among all other functions in $InitialSet_i$. Thus, as shown in Figure 4 for d_1, $Subset_1 = \{f_{1j}\}, j = 1, \ldots, 5$.

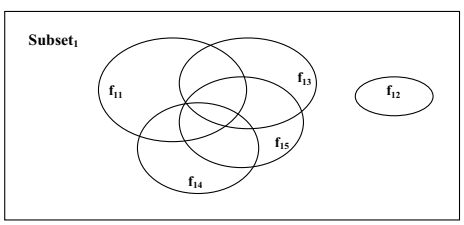


Fig. 4. *Example Subset for directive* d_1

3) *Although the aggregates in* $Subset_i$ *are not subset to any other aggregate, they can share common functionalities, which is an indicator of coupling.* Recall that a Capability is a self-contained functional abstraction that is minimally coupled with other Capabilities. We strive to minimize the coupling between abstractions by reducing their dependencies. Specifically, in the synthesis algorithm we use the abstraction level as an instructive factor in constructing minimally coupled aggregates. The technique of abstraction allows us to contain the dependencies within the boundaries of a higher abstraction. In particular, we identify aggregates that exhibit overlapping functionalities and aggregate them to form more decoupled abstractions. Hence, we create *aggregate subsets* $AG_{ij} (1 \leqslant i \leqslant n; 1 \leqslant j \leqslant m)$ from $Subset_i$ to contain aggregates with commonalities. Specifically,

$$AG_{ij} = \{f_{ix}, f_{iy} | f_{ix} \cap f_{iy} \neq \emptyset; 1 \leqslant x, y \leqslant m\}$$

such that $Subset_i = \bigcup_x f_{ix}, \forall f_{ix} \in AG_{ij}$;

We then abstract the entities of AG_{ij} to form higher level aggregates such that $AG_{ij} = \{F_{ij}\}$ where $F_{ij} = \{f_{ix} \cup f_{iy} \cup \ldots \cup f_{iz}\}; 1 \leqslant x, y, \ldots, z \leqslant m$. F_{ij} encompasses all aggregates in AG_{ij}. We term F_{ij} as **core functions**. Hence, we utilize core functions to derive and represent the functionality of system aggregates at a higher level of abstraction. For example, for directive d_1, in Figure 5, $AG_{11} = \{F_{11}\}$ where $F_{11} = \{\bigcup_j f_{1j}\}, j = 1, 3, 4, 5$ and $AG_{12} = \{F_{12}\}$ where $F_{12} = \{f_{12}\}$.

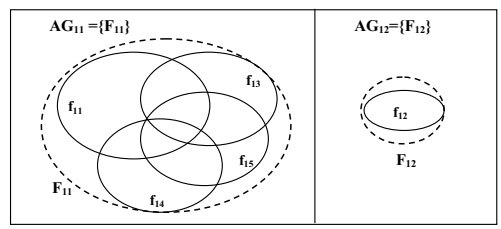


Fig. 5. *Example Aggregate Subsets for directive* d_1

4) *Let the core functions,* F_{ij}*, of all the aggregated subsets* AG_{ij} *related to directive* d_i *constitute the* i^{th} ***Core Function Set***, CFS_i*, such that* $CFS_i = \{F_{i1}, F_{i2}, \ldots, F_{ij}\}; 1 \leqslant i \leqslant n; 1 \leqslant j \leqslant m$. Hence, CFS_i comprises core functions that are functional abstractions initially defined at a more detailed level. These functional abstractions are potential Capabilities. Thus, as shown in Figure 6, $CFS_1 = \{F_{11}, F_{12}\}$.

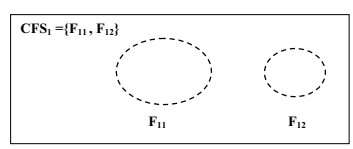


Fig. 6. *Example CFS for directive* d_1

Thus, in this manner, the synthesis algorithm defines *a Core Function Set (CFS) for each directive in the system.* Specifically, each directive d_i has an associated CFS_i. The elements of a CFS are core functions, which are aggregates derived from a systematic process of synthesizing directives.

Recall that Capabilities are functional abstractions that exhibit high cohesion and low coupling. Therefore, we now measure the cohesion and coupling values and examine the abstraction level of each core function in order to determine the set of Capabilities.

- *Cohesion*: For each directive the synthesis algorithm generates a CFS comprising core functions. The cohesion of

a core function is computed as an average of the relevance values of each participating directive in achieving that function. This implies that the list of directives associated with each core function in every CFS be enumerated; this necessitates substantial time and effort. Also, note that the core functions associated with different directives may be defined at various abstraction levels. Consequently, core functions may be subsets of one another resulting in redundant computations of relevance values. Furthermore, in our empirical analysis we observe that although the calculation of the average cohesion value is direct, the process of eliciting relevance values for each core function is highly cumbersome and notably subjective. These factors require us to explore alternate approaches for determining the cohesion of potential Capabilities.

- *Coupling*: Units are said to be coupled if changes in a source unit affect one or more dependent entities. The only information available for computing the coupling between the elements of CFSs in the synthesis algorithm is the set of common directives shared by the core functions. Experimental results show that determining coupling values merely based this number is unrepresentative of the actual implementation. Furthermore, the synthesis approach fails to provide information about the strength of dependency between functions. Hence, we conclude that the synthesis algorithm is ill-equipped to facilitate the computation of coupling between potential Capabilities.
- *Abstraction Level*: We know that each directive has an associated CFS whose elements are core functions. Empirical analysis reveals that at the abstraction level computed by the synthesis algorithm the core functions of a particular CFS do not share commonalities with other functions. However, any reduction in the abstraction level results in common intersections between aggregates. This is explained by the design of the synthesis algorithm, which terminates once a functional aggregate is established, as illustrated by the example of directive d_1. The synthesis algorithm indicates that the abstraction level of a core function is perhaps determined by examining its links with other core functions. Therefore, one needs to consider the abstraction level, and the links between aggregates when formulating Capabilities.

The synthesis algorithm attempts to identify Capabilities from the detailed directives of complex emergent systems. Given the large magnitude of these systems, considerable effort is required to establish the CFSs for 100s of directives. We note that, although the synthesis algorithm does provide insights regarding an ideal abstraction level of Capability, it is infeasible to automate the computation of cohesion and coupling measures. Therefore, it seems impractical that the synthesis algorithm be utilized for identifying Capabilities. This mandates that we design a more objective algorithm that is far less dependent on user input. Hence, we examine an alternative solution — a decomposition algorithm based on the top-down approach — in the following section.

IV. Decomposition

The decomposition algorithm utilizes a graph-based representation of user needs, *viz.* a Function Decomposition (FD) graph, to formulate Capabilities. An FD graph represents functional abstractions of the system obtained by the systematic decomposition of user needs. A need at the highest level of abstraction is the mission of the system and is represented by the root. We use the top-down philosophy to decompose the mission into functions at various levels of abstraction. We claim that a decomposition of needs is equivalent to a decomposition of functions because a need essentially represents some functionality of the system. Formally, we define an FD graph $G = (V, E)$ as an acyclic directed graph where V is the vertex set and E is the edge set. V represents the system functionality: leaves represent directives, the root symbolizes the mission, and internal nodes indicate system functions at various abstraction levels. Similarly, the edge set E comprises edges that depict decomposition, intersection or refinement relationship between nodes. These edges are illustrated in Figure 7. An edge between a parent and its child node represents functional decomposition and implies that the functionality of the child is a proper subset of the parents functionality. Only internal (non-leaf) nodes with an outdegree of at least two can have valid decomposition edges with their children. The refinement relation is used when there is a need to express a node's functionality with more clarity, say, by furnishing additional details. A node with an outdegree of one symbolizes this type of relationship with its child node. To indicate the commonalities between functions defined at the same level of abstraction the intersection edge is used. Hence, a child node with an indegree greater than one represents a functionality common to all its parent nodes. The FD graph utilizes these definitions to provide a structured top-down representation of system functionality, and thereby, facilitates the decomposition algorithm to formulate Capabilities in terms of their cohesion, coupling, and abstraction values. We discuss the mechanics of the algorithm next.

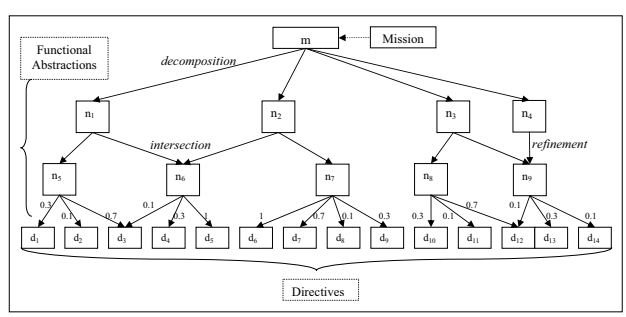


Fig. 7. *Example FD Graph* $G = (V, E)$

A. Algorithm

The input to the decomposition algorithm is an FD graph, $G = (V, E)$ that represents the functionality of the system

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

to be developed. We first determine the set of all valid combinations of internal nodes that can be considered as potential Capabilities. These combinations are termed $slices$. Then we compute the cohesion and coupling measures for each slice and examine the levels of abstraction to establish the finalized set of Capabilities.

We define slice S as a subset of V where the following constraints are satisfied:

1) *Complete Coverage of Directives:* We know that a Capability is associated with a set of directives, which is finally mapped to system requirement specifications (see Figure 1). Consequently, a set of Capabilities of the system has to encompass all the directives derived from user needs. The leaves of the FD graph constitute the set of all directives in a system. We ensure that each directive is accounted for by some Capability, by enforcing the constraint of complete coverage given by $\bigcup_{i=1}^{m} D_i = L$, where
   - D_i denotes the set of leaves associated with the ith node of slice S
   - $L = \{u \in V | outdegree(u) = 0\}$ denotes the set of all leaves of G
   - $m = |S|$
2) *Unique Membership for Directives:* In the context of directives, by ensuring that each directive is uniquely associated with exactly one Capability, we avoid implementing redundant functionality. Otherwise, the purpose of using slices to determine Capabilities as unique functional abstractions is defeated. We ensure the unique membership of directives by the constraint $\bigcap_{i=1}^{m} D_i = \{\phi\}$.
3) *System Mission is not a Capability:* The root is the high level mission of the system and cannot be considered as a Capability. The cardinality of a slice containing the root can only be one. This is because, including other nodes with the root in the same slice violates the second constraint. Hence, $\forall u \in S, indegree(u) \neq 0$.
4) *Directive is not a Capability:* A leaf represents a directive, which is a system characteristic. A slice that includes a leaf fails to define the system in terms of its functionality and focuses on describing low level details. Hence, $\forall u \in S, outdegree(u) \neq 0$.

The cohesion and coupling values for each slice is computed using the measures described next. We also discuss the average abstraction level of nodes that possess high cohesion and low coupling values.

- *Cohesion*: As in the synthesis algorithm, the cohesion of a node in a slice is computed as an average of the relevance values of the participating directive. The relevance values are assigned based on the values listed in Table I. However, we make a distinction between the parent and ancestor nodes of a directive. In order to reduce the need for user input, we elicit the relevance value of a directive only with respect to its parent node, whose cohesion is the arithmetic mean of the relevance values of its directives. Figure 7 illustrates relevance values of directives to their parents. However, the cohesion of an ancestor is computed as a weighted average of the size (number of associated directives) and cohesion of its non-leaf children. Specifically, the cohesion measure of an internal node n with $t > 1$ non-leaf children is:

$$Ch(n) = \frac{\sum_{i=1}^{t} (size(v_i).Ch(v_i))}{\sum_{i=1}^{t} size(v_i)}$$

  such that $(n, v_i) \in E$ and,

$$size(n) = \begin{cases} \sum_{i=1}^{t} size(v_i) & (n, v_i) \in E; outdegree(v_i) > 0; \\ 1 & outdegree(n) = 0 \end{cases}$$

- *Coupling:* To measure coupling we need information about dependencies between system functionalities. By the virtue of its construction, the structure of the FD graph represents the relations between different aggregates. In particular, we compute coupling between two nodes in a slice in terms of their directives. Two directives are said to be coupled if a change in one affects the other. We compute this effect as the probability that such a change occurs and propagates the shortest path ($dist$) between them. Note that the coupling measure is asymmetric. Generalizing, the coupling measure between any two internal nodes $p, q \in V$, where $outdegree(p) > 1, outdegree(q) > 1$ and $D_p \cap D_q = \{\phi\}$ is:

$$Cp(p, q) = \frac{\sum_{d_i \in D_p} \sum_{d_j \in D_q} Cp(d_i, d_j)}{|D_p|.|D_q|}$$

  where $Cp(d_i, d_j) = \frac{P(d_j)}{dist(d_i, d_j)}$ and $P(d_j) = \frac{1}{|D_q|}$.

  $P(d_j)$ is the probability that directive d_j changes among all other directives associated with the node q.
- *Abstraction Level:* The experimental results of the decomposition algorithm indicate that slices with nodes that exhibit maximum cohesion and decreased coupling are also at higher abstraction levels. We know that abstraction level is related to size; the higher the level of a node, the greater the number of its associated directives. Thus, the decomposition algorithm identifies Capabilities as nodes that exhibit high cohesion and low coupling but are also of increased sizes, which is undesirable from an implementation standpoint.

The decomposition algorithm provides an approach to automate the cohesion and coupling measures. Preliminary experimental results indicate that values computed using these metrics are indicative of desirable software engineering characteristics. In particular, we observe that on an average, in a slice, nodes *viz.* Capabilities that have high cohesion values also exhibit low coupling with other nodes. However, the decomposition approach fails to provide nodes at an abstraction

level that are optimal with respect to size. Therefore, we now explore a reconciliation between the synthesis and decomposition algorithms to determine Capabilities that are optimal with respect to the abstraction levels and the computations of cohesion and coupling.

V. Reconciliation

Sections III and IV describe the synthesis and decomposition algorithms to formulate Capabilities. In particular, we observe that the computation of coupling and cohesion values using the *decomposition approach* can be easily automated. This is because the coupling measure is a function of distance of change propagation and probability of change, and therefore, is completely objective. Likewise, the cohesion measure, although less objective, is conveniently computed for all functional abstractions. In contrast, the excessive subjectivity of the synthesis approach presents little scope for automating the formulation of Capabilities in complex emergent systems. However, unlike the decomposition algorithm, the *synthesis approach* provides insights about the optimum abstraction level of a Capability. Hence, we construct a composite algorithm to formulate Capabilities such that it incorporates elements of cohesion and coupling from the decomposition algorithm and that of the abstraction level from the synthesis algorithm. In this section, we first enumerate the steps of the composite algorithm. Then we use the example of the computer based library system to illustrate the reconciliation of the top-down and bottom-up approaches in the composite algorithm.

A. Composite Algorithm

The composite algorithm represents system functionalities at different abstraction levels using an FD graph, as in the decomposition approach. This is because, in the synthesis approach one needs to consider all possible system directives and then determine functional abstractions through an iterative process, which challenges the limited processing capacity of the human mind [21]. In contrast, the decomposition algorithm provides a more structured approach and begins with a single entity — system mission — that is easily comprehensible. Therefore, the input to the composite algorithm is an FD graph representation of the system functionality. The steps of the algorithm are detailed below:

1) Construct an FD graph to represent the system functionality.
2) Determine all possible slices from the FD graph. Each node within a slice is associated with a unique set of directives such that the union of these directive sets is equivalent to entire set of directives of a system.
3) Compute the cohesion and coupling values for nodes in each slice using the metrics defined by the decomposition approach, described in Section IV. Use these values to determine the average cohesion and coupling measures of a slice.
4) Similarly, compute the abstraction levels and sizes of each internal node in a slice.
5) The set of Capabilities is that slice which exhibits high cohesion, low coupling and comprises nodes of balanced abstraction levels.

We now illustrate the steps of the composite algorithm using the example of the library system.

1) Constructing FD Graph: In accordance with the first step of the composite algorithm an FD graph is constructed for the library system. To begin the process of systematic decomposition, we first identify the overall mission: develop software to automate library services. The mission is represented by the root node of the FD graph, which is illustrated in Figure 8. Observe that the mission is partitioned into functionalities of lower abstraction levels. The elliptical vertices in the FD graph denote the directives of the library system and the internal nodes indicate potential Capabilities. Also, the weight associated with an edge symbolizes the relevance value of a directive to its immediate parent node. Figure 8 shows these values as $1, 3, 7$, or 10. However, they are normalized on a [0,1] scale (as defined in Table I) for the computation of cohesion measure. Note that the weight of an edge between internal nodes is inconsequential and so is denoted by zero.

2) Determining Slices: In our experiment with the library system we computed 1014 valid slices from a possible 1048576 combinations of nodes. In essence there are six basic sets of slices and these are listed in Table II. Permutations of each basic slice set are also valid combinations. For example, for the slice S_1 of Table II, the permutations $\{n_1, n_2, n_3\}$ and $\{n_1, n_3, n_2\}$ are also considered as unique slices. This is primarily because the coupling measure is asymmetric, *i.e.* $Cp(n_i, n_j) \neq Cp(n_j, n_i)$, $i \neq j$; coupling is a function of probability of change which is computed using the size of a node. Consequently, permutations of the basic slice sets also need to be considered as individual slices. We conjecture that the coupling measure can assist in choosing an implementation order of Capabilities that potentially minimizes the impact of change.

TABLE II
Basic Slice Sets

Slice	Nodes	Permutations
S_1	n_1, n_2, n_3	2^3
S_2	n_1, n_3, n_4, n_5	2^4
S_3	n_1, n_3, n_4, n_8, n_9	2^5
S_4	n_2, n_3, n_{10}, n_{11}	2^4
S_5	$n_3, n_4, n_5, n_{10}, n_{11}$	2^5
S_6	$n_3, n_4, n_8, n_9, n_{10}, n_{11}$	2^6

An interesting observation from the FD graph of the library system is that nodes n_6 and n_7 are the only nodes that are not a part of any slice. Further analysis reveals the following explanation: Nodes n_6 and n_7 are internal nodes with only directives as their siblings. Moreover, the parents of n_6 and n_7 are internal nodes whose children are directives and internal nodes. The constraint of a slice definition, *viz.* unique membership of directives, disallows n_6 and n_8 or n_7 and n_3 from being a part of the same slice. In addition, the

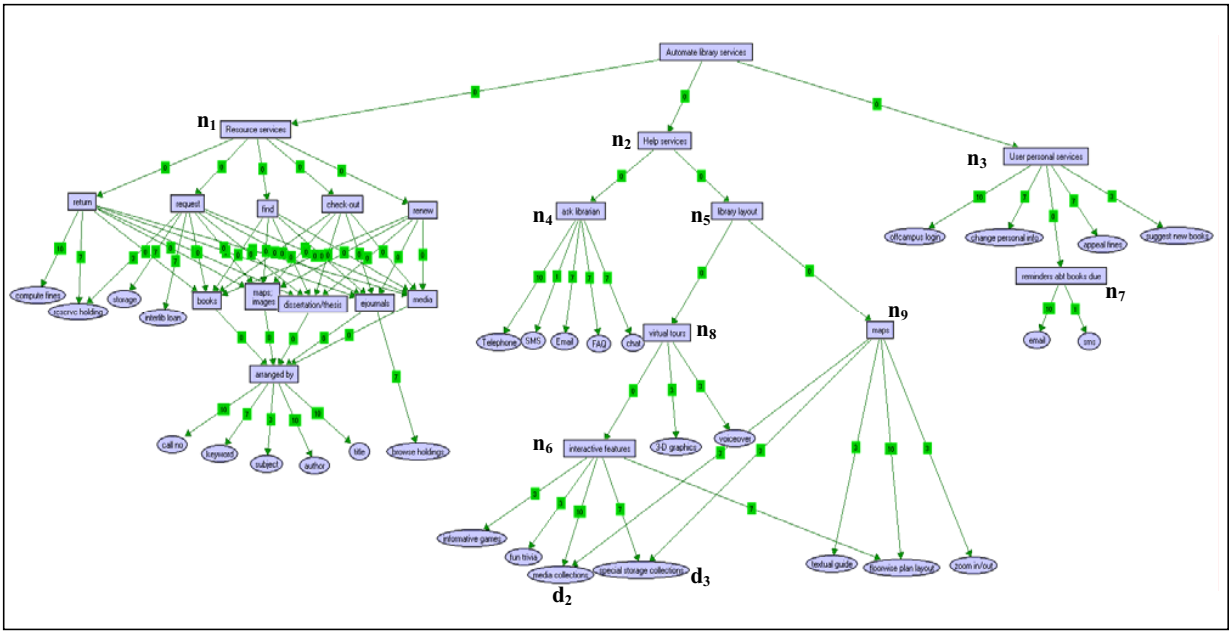


Fig. 8. *FD Graph of Library System*

requirement for complete coverage of directives of nodes in a slice necessitates the exclusion of n_6 and n_7 from any slice. However, it is possible that n_6 or n_7 can be the root of a large subgraph, in which case their exclusion is detrimental to the formation of Capabilities. This observation compels one to explore the relationship and distribution of internal nodes and directives, when defined at the same level.

3) Computations: Once the slices are determined we compute the cohesion, coupling and abstraction values for each node of a slice. The arithmetic mean of these values provides statistics that help ascertain the quality of a slice from a software engineering perspective. Before discussing the computations, however, we first analyze the relationship between a node's abstraction level, its depth and its size. This is because the objective of the composite algorithm is to formulate Capabilities that not only exhibit high cohesion and low coupling, but are also of balanced abstraction levels. Furthermore, unlike abstraction level, cohesion and coupling are well established concepts that are accepted by the software engineering community. Therefore, it is imperative that we present our notion of an "abstraction level" using insights provided by the synthesis approach and discuss its role in identifying Capabilities.

- *Abstraction Level, Depth and Size:* An abstraction presents information essential to a particular purpose by omitting irrelevant elements. A Capability is a functional aggregate that indicates the functionality expected of the system from a high-level perspective while ignoring minute details. Similarly, the mission of a system is a functional aggregate described at the highest level of abstraction because it states the overall system functionality *sans* low-level information. Therefore, with respect to the FD graph, the root is of the highest abstraction and a directive is the lowest. Furthermore, we observe that the abstraction of the internal nodes decreases with their increased distance from the root; distance is the length of the shortest path from the root. We refer to this distance as the *depth* of a node. More specifically, the greater a node's depth, the lower is its abstraction level. Therefore, for a node $n \in V$ in an FD Graph G, the qualitative relation between abstraction level and depth is denoted as:

$$Abstraction(n) \propto \frac{1}{depth(n)}$$

We now discuss the relation between a node's depth and its size. Recall that size is the number of directives associated with a node. The FD graph of the library system indicates that the size of an internal node decreases as its depth increases. For example, in Figure 8 node n_2 has size 14 and depth 1 whereas n_4 is of size 5 and depth 2. We confirm if the random variables, size and depth, are truly correlated by using a scatter plot. Specifically, we plot the average values of size and depth of a node within a slice. This data is obtained from the 1014 slices computed from the FD graph of the library system. The scatter plot is shown in Figure 9 where, within a given slice, the average size of a node is plotted against the average depth of a node. We discuss the following observations about the scatter plot diagram:

a) Permutations: In the scatter plot of Figure 9, each data point with coordinates $(size, depth)$ corresponds to the basic slice set listed in Table II, and therefore, is denoted

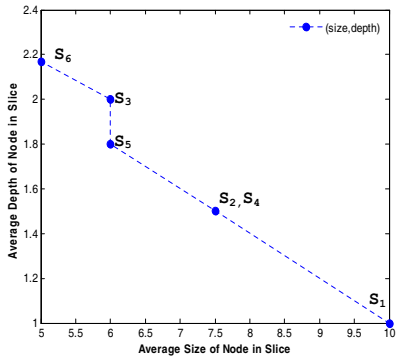


Fig. 9. *Illustrating the relation between depth and size*

by S_i. Although we plot the average size and the depth of a node in each of the 1014 slices computed from the FD graph of the library system, there are only five data points. This is because the permutations of the basic slices results in additional valid slices. However, the average node size and depth values remain unchanged for the permutations of the same basic slice set. Therefore, each point $S_i, (i = 1, \ldots, 6)$ represents the average $(size, depth)$ of a node, which is unchanging in the permutations corresponding to the basic slice sets, shown in Table II.

b) Correlation: The scatter plot shows that there is a relation between the depth and size, which is reaffirmed by the large value of their correlation coefficient $Corr(size, depth) = -0.966630075$. This implies that depth and size are highly negatively correlated. In addition, as discussed earlier, we know that the level of abstraction decreases with an increase in depth. Using this relation and the negative correlation between size and depth, one infers that the size of a node is proportional to its abstraction level. Therefore, for a node $n \in V$ in an FD graph $G = (V, E)$, the qualitative relation between abstraction level and size is denoted as:

$$Abstraction(n) \propto size(n)$$

Hence, the relation between a node's size, depth and abstraction level is used to assist in judicious identification of Capabilities. In particular, we conclude that a balanced abstraction level is influenced by the size of a node.

- *Cohesion and Coupling:* The cohesion and coupling measures are computed using the metrics defined in Section IV. The maximum, minimum, and median average cohesion and coupling values of the slices in the library system are also detailed in Table III, along with the average size and depth values.

4) Selecting Capabilities: We illustrate the final step of the composite algorithm — determining the optimum set of Capabilities from the set of all slices — using the example library system. Of specific interest are slices that exhibit, on an average, high cohesion and low coupling values. In particular, among all slices computed from the FD graph (Figure 8) of the library system, we examine the slices, PS_1 and PS_2, that have the two highest cohesion values. PS_1 and PS_2 are permutations of the basic slice sets S_1 and S_2 respectively. The cohesion and coupling values of slices PS_1 and PS_2 are presented in Table IV. Note that the cohesion and coupling values of PS_1 and PS_2 are higher and lower respectively, than the median values of the slices of the library system, described in Table III. In particular, slice PS_1 has maximum cohesion and minimum coupling values. Slice PS_2 exhibits the second highest cohesion value and a coupling measure of 1.09577 that is lower than the overall coupling median of 2.76445.

TABLE III

Average Values of Slices of Library System

RANGE	COHESION	COUPLING	SIZE	DEPTH
Maximum	0.65119	3.58603	10	2.16667
Minimum	0.599048	0.547891	5	1
Median	0.599048	2.76445	5	2.16667

TABLE IV

Average values of Slices S_1, S_2 and S_3

SLICE	COHESION	COUPLING	NODES
PS_1	0.65119	0.547891	$\{n_3, n_1, n_2\}$
PS_2	0.636873	1.09577	$\{n_4, n_3, n_1, n_5\}$
PS_3	0.603689	1.59961	$\{n_4, n_9, n_3, n_8, n_1\}$

According to the decomposition algorithm, slice PS_1 is the most optimal among all slices of the library system. This is because it exhibits maximum cohesion and minimum coupling values when compared to all other slices. In contrast, the synthesis algorithm chooses slice PS_2 as the desirable set of Capabilities. Recall that the synthesis approach emphasizes on low abstraction levels, and consequently, reduced sizes. The average size of each node in PS_1 is 10 where as that of PS_2 is 7.5 as shown in Table V. The implementation size of individual aggregates in PS_1 can be reduced if node n_2 is replaced by n_4 and n_5, which results in a combination of nodes (n_1, n_4, n_5, n_3), *viz.* basic slice set S_2 and in particular, the least coupled permutation, PS_2. This also implies that nodes n_4 and n_5 are at a lower level of abstraction than node n_2. The marginal increase in coupling of PS_2 is offset by the advantage of constructing smaller sized Capabilities. We observe from the FD graph in Figure 8 that there are no intersection edges between nodes n_4 and n_5, signifying that this is a balanced abstraction level. In contrast, let us consider the scenario where we choose to implement nodes, say n_8 and n_9 instead of n_5, which are defined at much lower level of abstraction and are of a smaller size. The node combination $(n_1, n_4, n_8, n_9, n_3)$ is actually the basic slice set S_3 listed in Table II. We choose the permutation PS_3=$\{n_4, n_9, n_3, n_8, n_1\}$ because it has the lowest coupling among all possible permutations of S_3. Recall that all permutations exhibit the same cohesion, and hence, fails to influence the selection. The cohesion, coupling, size and depth values values are shown in comparison with PS_1 and PS_2 in Table

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

IV and Table V. Slice PS_3 has a lower cohesion and a higher coupling value than PS_1 and PS_2. However, it exhibits values that is better than the median cohesion and coupling values of the overall slices of the library system. In addition, PS_3 also has a smaller size when compared to the selections of the decomposition algorithm — PS_1— or the synthesis algorithm — PS_2. Collectively, these factors seem to indicate that this slice is perhaps a more optimal choice than either PS_1 or PS_2 — as illustrated by Table V. However, we observe from the FD graph in Figure 8 that nodes n_8 and n_9 share common directives d_2 and d_3. If these nodes are considered as Capabilities, then directives d_2 and d_3 will be associated with only one Capability. This implies that linkages will be broken to ensure that each directive is bound to either n_8 or n_9. Even so, this does not eliminate the implicit coupling that exists between n_8 and n_9. Consequently, it is difficult to consider nodes n_8 and n_9 as Capabilities with reduced coupling. This nonconformity with a basic tenet of the Capability definition forces us to reject PS_3 as an optimal set, and instead, choose slice PS_2.

TABLE V

Average Size and Depth

SLICE	AVERAGE SIZE	AVERAGE DEPTH
PS_1	10	1
PS_2	7.5	1.5
PS_3	6	2

To summarize, the composite algorithm constructs an FD graph, determines all possible slices and utilizes the cohesion and coupling computations of the decomposition algorithm to produce potential sets of Capabilities. Then, it incorporates the criteria of a balanced abstraction level and reduced sizes as illustrated by the synthesis algorithm to choose an optimal slice for the library system. Thus, the elements of cohesion and coupling from the top-down approach and the abstraction level from the bottom-up approach constitute the composite algorithm to determine Capability sets for developing change-tolerant systems.

VI. CONCLUSION

Software engineering methods for system development are often based on a top-down or a bottom-up approach. However, our solution framework, CE, constructs complex emergent systems as change-tolerant entities by utilizing a complementary approach. In particular, we use a composite algorithm to formulate Capabilities as maximally cohesive and minimally coupled functional abstractions of a system. Presently, a Capability depicts only the functionality of a system; integrating non-functional aspects into the definition of a Capability is part of future work. The cohesion and coupling measures of these basic building blocks are computed as in the decomposition algorithm and the abstraction level as defined by the synthesis algorithm. Note that the former algorithm is based on a top-down approach while the latter, on a bottom-up approach. Thus, the composite algorithm is a blend of the two polar approaches. Experimental results further substantiate the need for such a complementary approach. Our experience in assessing the efficacy of the synthesis and decomposition algorithms aids in understanding the essence of a Capability, and emphasizes that the design of Capabilities is in fact a reconciliation of diametrically opposite approaches to problem solving.

REFERENCES

[1] F. Heylighen, "Self-organization, emergence and the architecture of complexity," in *Proceedings of the 1st European Conference on System Science*, 1989, pp. 23–32.

[2] D. L. Parnas, P. Clements, and D. Weiss, "The modular structure of complex systems," in *Proc. of 7th International Conference on Software Engineering*, Mar. 1984, pp. 408–417.

[3] R. Ravichandar, J. D. Arthur, and S. Bohner, "Capabilities engineering: Constructing change-tolerant systems," in *40th Hawaii International Conference on System Sciences (to appear)*, 2007.

[4] M. M. Lehman, "Laws of software evolution revisited," in *Proceedings of the 5th European Workshop on Software Process Technology*, Nancy, France, Oct. 1996, pp. 108–124.

[5] S. D. P. Harker, K. D. Eason, and J. E. Dobson, "The change and evolution of requirements as a challenge to the practice of software engineering," in *Proceedings of IEEE International Symposium on Requirements Engineering*, Jan. 1993, pp. 266–272.

[6] Y. Malaiya and J. Denton, "Requirements volatility and defect density," in *10th International Symposium on Software Reliability Engineering*, Los Alamitos, California, Nov. 1999, pp. 285–294.

[7] D. Zowghi and N. Nurmuliani, "A study of the impact of requirements volatility on software project performance," in *9th Asia-Pacific Software Engineering Conference*, Gold Coast, Australia, Dec. 2002, p. 3.

[8] G. A. Tull, "Guide for the preparation and use of performance specifications," Department Of Defense," Headquarters U.S. Army Material Command, 1999.

[9] D. of Defense. (2000) Guidebook for performance-based services acquisition (pbsa) in the department of defense. Acquisition, Technology, and Logistics Web. [Online]. Available: www.acq.osd.mil/dpap/Docs/pbsaguide010201.pdf

[10] M. Montroll and E. McDermott. (2003) Capability-based acquisition in the missile defense agency: Independent research project. Storming Media. [Online]. Available: http://www.stormingmedia.us/82/8242/A824224.html

[11] D. Rowe and J. Leaney, "Evaluating evolvability of computer based systems architectures-an ontological approach," in *International Conference and Workshop on Engineering of Computer-Based Systems*, Mar. 1997, pp. 360–367.

[12] D. L. Parnas, "On the criteria to be used in decomposing systems into modules," *Communications of the ACM*, vol. 15, pp. 1053–1058, 1972.

[13] E. Yourdon and L. Constantine, *Structured Design*. Englewood Cliffs, N.J: Prentice-Hall, 1979.

[14] W. P. Stevens, G. J. Myers, and L. L. Constantine, "Structured design," *IBM Systems Journal*, vol. 13, no. 2, pp. 115–139, 1974.

[15] J. A. Goguen and C. Linde, "Techniques for requirements elicitation," in *Proc. Int. Symp. Req. Engineering*, Los Alamitos, California, 1993, pp. 152–164.

[16] L. B. S. Racoon, "The complexity gap," *ACM SIGSOFT Software Engineering Notes*, vol. 20, pp. 37–44, 1995.

[17] P. Zave and M. Jackson, "Four dark corners of requirements engineering," *ACM Transactions on Software Engineering and Methodology*, vol. 6, pp. 1–30, 1996.

[18] S. Lauesen and O. Vinter, "Preventing requirements defects: An experiment in process improvement," *Requirements Engineering*, vol. 6, pp. 37–50, 2001.

[19] J. M. Bieman and L. M. Ott, "Measuring functional cohesion," *IEEE Transactions in Software Engineering*, vol. 20, pp. 644–657, 1994.

[20] B. W. Boehm, *Software Risk Management*. New York: IEEE Computer Society Press, 1989.

[21] G. A. Miller, "The magical number seven, plus or minus two: Some limits on our capacity for processing information," *The Psychological Review*, vol. 63, pp. 81–97, 1956.

Model-based Data Processing with Transient Model Extensions

Michael Thonhauser[1,2], Gernot Schmoelzer[1,2], Christian Kreiner[1,2]

[1]Institute for Technical Informatics
Graz University of Technology, Austria
Inffeldgasse 16, A-8010 Graz
michael.thonhauser@TUGraz.at

[2]Salomon Automation GmbH
Friesach bei Graz
A-8114 Friesach, Austria

Abstract

Software is often constructed using a layered approach to encapsulate the functionality in different layers. Individual requirements of each layer demand layer specific data structures. These data structures typically provide redundant information with respect to the data source.

Providing a Model Driven Software Development approach for creating these data structures leads to overlapping data models, each containing data structures defined by the data source. Because putting all various requirements of the software layers in a single data model can lead to difficulties, each software layer should only extend the basic data source model with its specifically needed model elements.

This paper presents a mechanism for transient extension of a data model. Using this mechanism, a basic data model can be used by every layer, being extended by additional attributes and classes for satisfying layer specific requirements.

Keywords: Model-driven development; Data modeling

1. Introduction

Data-intensive systems are characterized by manipulating and displaying a large amount of structured data. This data is saved in some persistent storage such as a database or a file. Such data-intensive systems are often designed in three layers [6]. including *presentation layer*, *domain* (or business function) *layer* and *data source layer*.

Information contained in the data source is accessed through the data source layer. This layer transfers data contained in persistent data structures such as database tables, into dynamic data structures, such as arrays or object composites. The transfer functions can be coded manually or can be generated using a data model of the persistent data[1].

Because the problems to be solved by data-intensive software are quite complex, manual programming of transfer functions can quickly lead to logical errors. Bran Selic describes this problem in [14], distinguishing between essential and accidental complexity of software. Essential complexity is inseparable from the problem, but accidental complexity is a direct consequence of the chosen problem solution. For reducing accidental complexity the problem should be looked at an abstract viewpoint, ignoring program language specific details.

This abstraction can be provided by a plattform independent model (PIM), which is created during a Model Driven Development (MDD) process. While MDD tries to capture all aspects of a program using different variants of modeling, like activity diagrams or class diagrams, agile model driven development (AMDD) has its focus on the aspect of a model currently needed. In data-intensive application development one of the first modeled aspects of each layer is its data structure.

2. Motivation

2.1. Layered software

In the previous section the idea of software layers has been introduced. Each software layer internally uses data structures, holding the needed data of this layer. Fig. 1 shows this data structures and depicts the dependencies between them. Every data structure can be seen as a layer specific view on the information generally available to the program. Data structure $S1$ holds the transformed persistent data, which is compliant to the structure of the data source.

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

$S2$ repesents the data structure used by the domain layer and $S3$ is contained in the presentation layer.

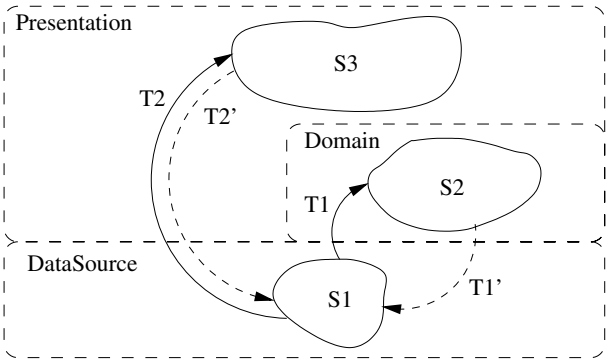


Figure 1. Data structures in software layers

Fig. 2 illustrates an example use case from the business domain of logistics. Boxes containing small items are handled by a warehouse management system (WMS). The WMS is used for checking the amount of items contained in the box, before the box leaves the warehouse. The user interface consists of a table, which additionally displays the number of dozens and single items in a box, to ease the quality check of the boxes. It also makes use of a business function calculating the weight of each box for issuing an alert, if the weight of all boxes reaches the weight limit for transportation.

This example illustrates different requirements for data structures in various application layers. Requirements for $S1$ are the reduction of duplicate information, thus providing support for *data normalization* [1]. This reduction is needed for optimal storage usage, efficient data transfer and to avoid inconistency of the stored data. Also the data structure should be easily mappable to the persistent data source.

$S2$ is driven by requirements for *simplifying complexity* of the application logic, adding *transaction handling* and *data consistency checks*. This aims at avoidance of inconsistent data and is done through *adding data redundancy*.

$S3$ contains the data displayed in the presentation layer. Its content meets user requirements and *usability aspects*, which often requires displaying additional information.

As illustraded in Fig. 1 there exist several dependencies of the data structures in the different layers. Because $S1$ holds all the information available to the application, additional data in $S2$ and $S3$ is normally related to data contained in $S1$. This requirement can be fulfilled by extending $S1$ with a transformation $T1$ or $T2$.

The way this transformation is performed depends on the abstraction level used for application development.

- The WMS described in Fig. 2 can be implemented using a relational database and a structured programming language like C. This programming language can work with a database manipulation language (DML) like SQL to define the data structures $S1$, $S2$ and $S3$. In this case the transformation $T1$ and $T2$ of the data structures is done by extending the DML statement.

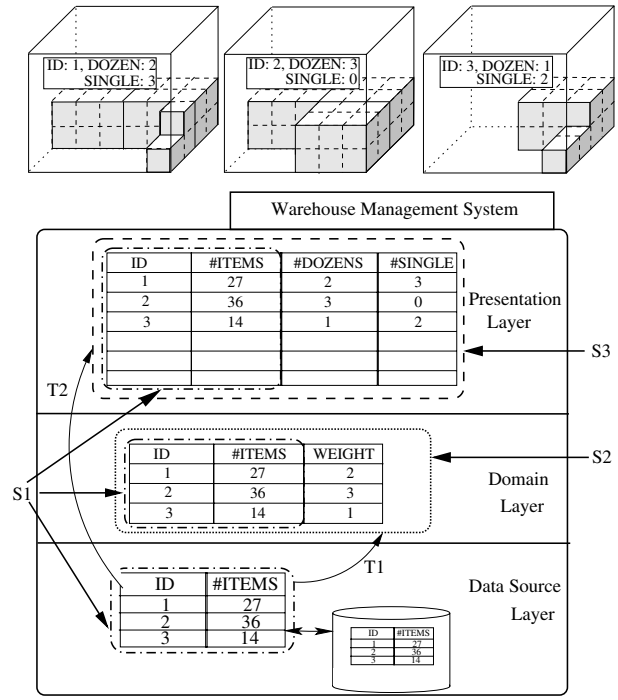


Figure 2. Warehouse management example

- A higher level of abstractions is provided by object oriented languages combined with object-relational methods [6]. This solution enables type safety for $S1$, $S2$ and $S3$ consisting of objects. The transformation is done with additional source code for the implemenation of $S2$ and $S3$.

- A further abstraction level is reached by using a MDD approach. This leads to a correspondend data model $DM1$ for $S1$, $DM2$ for $S2$ and $DM3$ for $S3$. Following the dependencies of the different data structures, the dependencies of the data models can be seen as $DM1 \subseteq DM2$ and $DM1 \subseteq DM3$. In this case transformation $T1$ or $T2$ is done by extending $DM1$. This extension adds information about additional data needed in the presentation layer leading to $DM3$ and to $DM2$ respectivly in the domain layer.

  We call this transformation mechanism Transient Model Extension (TME). Since $DM1$ is used as the basic model for this mechanism, $DM2$ and $DM3$ does not need to be stored in the corresponding layers. Only

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

the transformation rules needed for construction of the appropriate data model need to be saved instead.

2.2. Model Driven Development

Model Driven Development (MDD) [9] is an approach to implement a software system by describing it in a Platform Independend Model (PIM). A PIM defines associations between data and behavior of the software and it is used as input for generators producing a platform specific model (PSM). To support MDD the Object Management Group (OMG) has released the Model-Driven Architecture (MDA) containing standards allowing specification and transformation of models. Another dominant approach to MDD are Software Factories, which are proposed by Microsoft and can be seen as a new software development paradigm. Differences of these two approaches are discussed in [4].

Data modeling

Models of software design are often specified using Unified Modeling Language (UML) [11], another standard of the OMG. UML models are based on a metamodel and are situated in the user model layer of the four-level metamodel hierarchy [10]. UML describes several diagrams, which can be used to model different aspects of a software. Structural and behavioral diagrams are differentiated. One example of a structural diagram is the class diagram. It is used to model the structure of classes, such as attribute and methods, as well as the association between the different classes in the model.

For data modeling purposes the metamodel of a class diagram can be extended focusing only on class attributes and associations [1].

Agile Data Modeling relies on iterative construction of data models, where each data model satisfies the requirements needed in the current iteration. It is best suitable for applications, that rely on relational databases for persistent data storage. Agile Model Driven Development (AMDD) also uses an iterative approach, instead of extensive models being generated in the normal MDD process.

3. Transient Model Extension

The structure of a data model can be based on the four-level metamodel hierarchy of the OMG, whereas level M3 and level M2 are the same for each data model. Model level M1 contains the domain specific data model, holding the domain specific classes and associations. This elements are used for construction of objects which are contained in model level M0. Generally speaking every model can be seen as an instance of the model in the lower layer. Considering this fact, it is obvious that changing the model in level M1 results in additional data contained in level M0.

Fig. 3 shows the four-level metamodel hierarchy with respect to the data structures defined in Fig. 1. According to this figure $DM1$ is part of every data model used by the different software layers. Data model extensions (DME) correspond to the defined mappings $T1$ and $T2$ in Fig. 1.

$DM2$ is the data model used by the domain layer. It is defined as a combination of $DM1$ and $DME2$. In the same way the data model of the presentation layer ($DM3$) results from combining $DM1$ with $DME3$.

Applying the AMDD approach to a layered software architecture results in the definition of the data model describing the data structures used by the data source layer. In the next steps the corresponding data models for the domain and the presentation layer are defined. According to Fig. 3 a separated definition of each data model used by the different software layers is not feasible. Identical information needed by different layers is duplicated in the models, leading to maintenance difficulties of the shared information. Also each layer needs to store its own data model, which can lead to increasing storage requirements by the application. To overcome these problems a data model for all layers could be used, that is adapted for the requirements of the different layers.

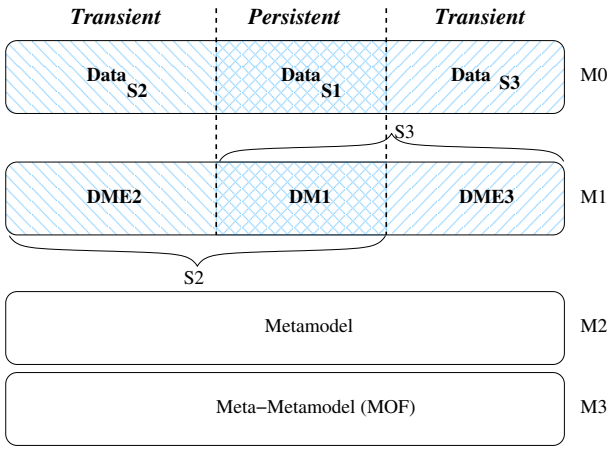


Figure 3. Data model structure

3.1. Transient Model Extension use cases

To get a model for a specific software layer other than the data source layer, its data model normally needs to be extended, which is done on the fly. We call this mechanism Transient Model Extension (TME). It is illustrated by arrows $T1$ and $T2$ between the data structures in Fig. 1 and by fasciated areas in Fig.3.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Like the extensions of a data structure implemented in an object-oriented language follows the rules of the language compiler, extensions of the model have to be compliant to its metamodel.

Typical TME use cases are:

Attribute Extension. Extending a class with additional attributes is done to model additional information, that belongs to this class and can be computed from other class attributes or associated classes. An extended attribute needs to be of a type already defined in the model.

Additional Associations. Requirements of an upper software layer can require additional associations between classes, that are not associated in the data source. Usually this is done for simplifying associations of related classes in the model. Consider a class A which has a one-to-many association to class B. Class B in turn has a many-to-one association to class C, which has an association with a class D. If now an object of class A needs the associated objects of class D, it has to traverse two associations. This situation can be simplified by directly associationg class A with class D.

Class Extension. Extending the model with additional classes is a combination off all previous use cases. Adding additional classes can be done to meet the requirements for data redundancy or to simplify the functions in the domain layer.

The *data gathering mechanism* for the extended model elements is specified in the TME process. Calculation of model element values can be performed with different mechanisms.

The first approach is the use of a *hook* function, which is called everytime the value of the extended element is requested. The other alternative implements the *observer* pattern [7], implying changes to the extended model element only when one of the observed model elements changes.

One important aspect of the TME approach is the temporary extension of a data model. Because the same data model of the data source layer is used as basis for the domain and presentation layers, extensions applied by these layers should not affect the (data) models of other layers.

3.2. Generic Components and TME

Generic components can be seen as software components for which specific properties have been left variable during components implementation [2]. For AMDD it means that component functionality is based on a data model as well as a metamodel and the model is used for configuration.

In case of generic components TME can be used to configure the runtime behavior of the component. Imagine a user interface component like a table, which dynamically displays data of a given class in the associated data model. Extending this class with additional attributes leads to displaying of additional table columns.

4. Implementational considerations

A more detailed data model of the warehouse management system (WMS) use case presented in Sect. 2 is used for demonstration of the TME mechanism. Although this example has a higher degree of complexity, we note that this example is still a simplified version of our real world application.

Provided that the structure of the persistent data has been modeled, an **Entity Container** (EC) [13] can be used as a model-based object oriented data cache. The architecture of the EC is shown in Fig. 4. The EC provides distinguishable objects called entities, identified by a unique value. It operates on two levels of the four-level metamodel hierarchy of OMG implementing the instance of relation between the two levels.

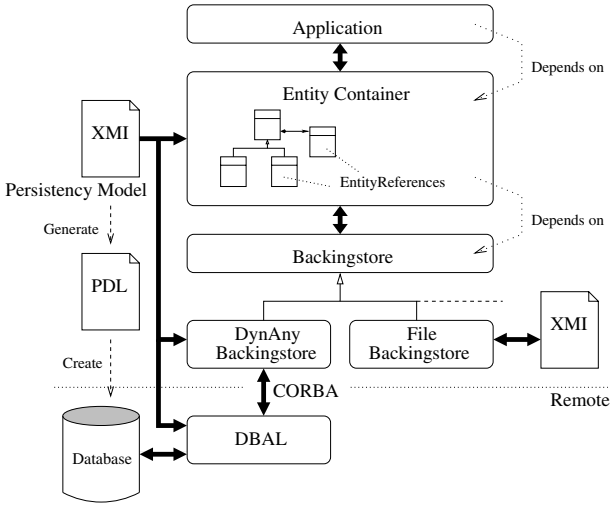


Figure 4. Entity container architecture

The UML data model is stored in a file using the XML Metadata Interchange (XMI) format. This file contains the UML model of the persistent data and is used by the EC and the associated backingstore, such as an object-relational bridge (DBAL). Data entities in the EC are accessed using a dynamic interface.

In our MDD approach the database is created from the persistency model, which is also used to configure the EC and its associated backingstore.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

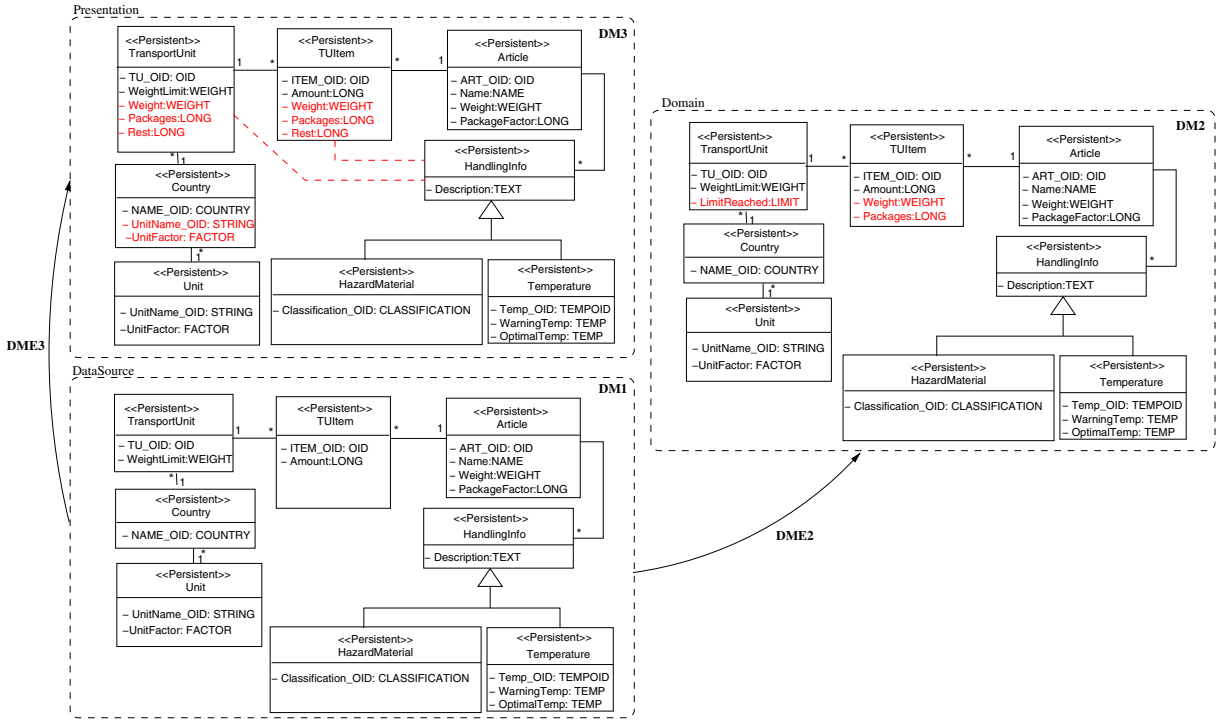


Figure 5. Data models used by software layers for extended example

4.1. Extended example

Instead of the data structure shown in Fig. 2, the detailed data structure allows a mixture of items from different articles in the same box. Each article defines a weight and can be associated with additional handling informations like its preferred temperature or its hazard class. Additionally each box is associated with a target country, which has a preference for a unit system.

Fig. 5 shows the data models of the different structures used in the application. $DM1$ defines the data structure $S1$ of the data source layer, which is used for generating a database by applying the previously described MDD approach. In the persistent data storage only minimal information is saved about the items and boxes, which are modeled as class `TransportUnit`.

The following use cases need to be fulfilled by the different layers of the warehouse management system for goods issue:

UseCase I: It is required to calculate the weight of all items placed on a transport unit. The function can be used to inform a user whether a transport unit is overweight, which can be caused by the individual weight of the different articles.

UseCase II: A user needs a control for displaying the number of items on a transport unit. As stated in section 2 for better usability the displayed item count is split up in packaged items and rest items. Note that every article has an individual factor for calculating the number of packaged items and rest items.

Additionally, a user should also be able to control the weight of the transport unit. The weight is displayed in the preferred unit system of the the inspected transport unit's target country.

The WMS is implemented using the previously described software layers. Because the persistent data model is used for generating the database, it defines the model for the data source layer. All application layers mentioned above are based on the data from the data source of the WMS.

To meet the requirements of the previously defined use cases $DM1$ in Fig. 5 needs to be extended applying the TME mechanism, which is illustrated by the arrows labeled $DME2$ and $DME3$. UseCase I is implemented with the additional attributes contained in $DM2$, while UseCase II is realized by additional attributes and associations in $DM3$.

For $DM2$ the value of the additional weight attribute in `TUItem` is calculated using:

$$Weight_{TUItem} = Amount_{TUItem} * Weight_{Article} \quad (1)$$

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

The value of this attribute is used in the calculation of the weight attribute of `TransportUnit`, which is based on equation:

$$Weight_{TU} = \sum_{TUItems} Weight_{TUItem} * UnitFactor_{Country} \quad (2)$$

For UseCase II the value of the package and rest attribute of `TUItem` is calculated using the *PackageFactor* attribute of the associated `Article` and the *amount* attribute of `TUItem`.

Keeping these facts in mind the following situation can be thought of. There can exist some articles, which also have a heavy-weight packaging. If this fact needs to be considered in the WMS, the following modifications have to be made.

First a new attribute containing a package weight needs to be added to `TUItem` in $DM1$. Additionally the attribute holding the package count and its corresponding calculation function need to be added to `TUItem` in the TME process. Also Eq. 1 has to be changed to

$$Weight_{TUItem} = Amount_{TUItem} * Weight_{Article} + Packages_{TUItem} * PackageWeight_{TUItem} \quad (3)$$

With this additional extension one benefit of the TME method is demonstrated. Because Eq. 2 relies only on the value of the weight attribute, changes in the Eq. 1 calculating the value of this attribute are transparent to equation Eq. 2.

4.2. TME with the Entity Container

Listing 1 illustrates the code for the data source layer. This layer holds the data model of the application. It also provides access to the data source. According to Fig.4 each EC uses a data model and a backingstore, which holds the connection to the data source. The parameter for initialization of the EC in the upper layers of the application are retrieved from the data layer.

Listing 1. Data source layer

```
..
        // Datastructure holding the data model
    private IModel dataModel_;

    // Backingstore connecting to the database
    private IBackingStore bs_;

        /**
        * Public constructor initializing data layer
    **/
    public DataLayer() {
        // initialize the data model
        dataModel_ = Model.create4TME("resource/WMSModel.xml.zip");
        // open connection to database for backingstore
        IPersistency persist = new RemoteDbalPersistency();
                // initialize backingstore
        bs_ = new JavaDynAnyBackingStore(dataModel_, persist);
    }

    //Method for retrieving the model of the data source layer
    public IModel getDataModel() {
        return dataModel_;
    }

    // Method for getting access to the data source
    public IBackingStore getBS() {
        return bs_;
    }
..
```

Listing 2 demonstrates the implementation of the business function for checking the current weight of an item. This method is invoked by providing the ID of the current transport unit. In the initialization phase of this method the data model of the data source layer is retrieved. This model is extended with an additional attribute, containing the weight of the transport unit item. A new EC is instantiated, which is based on the extended model and the backingstore contained in the data source layer.

In this example we demonstrate, that the value for the additional attribute is provided using a hook function.

Listing 2. Implementation of UseCase I

```
  // business function calculating weight limit
 public boolean weightLimitReached(String tuID){
   DataLayer dl = Application.getDataLayer();
   // get persistent data model
   IModel layerModel = dl.getDataModel();
   //apply TME for attribute weight of class TUItem
   IAttribute weight = layerModel.TME_addNewAttribute(ITEM, attr_Weight, WEIGHT);
   weight.setHook(new ItemWeightHook());
   IAttribute limit = layerModel.TME_addNewAttribute(TU, attr_Limit, Limit);
   limit.setHook(new LimitHook());

   //initialize new backingstore EC
   IBackingStore bs = dl.getBS();
   BackingStoreEntityContainer ec_ = new BackingStoreEntityContainer(
                                    layerModel, bs);
   // load transport units from datasource
   ec_.load(TransportUnit);

   //lookup specific transportUnit
   ITemplate template = new Template();
   template.setAttribute(TU, tuID);
   IEntityReference ref = ec_.findByPrimaryKey(TransportUnit, template);

   // return value of the limit attribute for selected transport unit
   return ref.getAttribute(attr_Limit)
 }
 ..

private class ItemWeightHook extends EntityAttributeHook {
  // called for attribute Weight in class TUItem
  public IValue getAttribute(IEntityReference entity, String name) {
   double retVal = 0;
   ICollection<IEntityReference> articles = entity.getConnections(ARTICLE);

   // calculate weight of associated attributes
   for (IEntityReference article: articles)
        retVal += ((DoubleValue)article.getAttribute("Weight")).getValue()*
   ((DoubleValue)entity.getAttribute("Amount")).getValue();

   return new DoubleValue(retVal);
  }
 ..

private class LimitHook extends EntityAttributeHook {
  // calculates the value of attribute limit in TUItem
  public IValue getAttribute(IEntityReference entity, String name) {
   double currentWeight = 0;
   ICollection<IEntityReference> items = entity.getConnections(ITEM);

   for (IEntityReference item: items)
        currentWeight += ((DoubleValue)item.getAttribute("Weight")).getValue();
   boolean ret = (entity.getAttribute("limit").compareTo(currentWeight) >= 0);

   return new BooleanValue(ret);
  }
}
```

Listing 3 shows the implementation of the modified hook method implementing Eq. 3. This implementation can be applied to the weight attribute in Listing 2.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Listing 3. Changed hook implementing Eq.3

```
private class ItemWeightHook extends EntityAttributeHook {

  public IValue getAttribute(IEntityReference entity,String name) {
   double retVal = 0;
   ICollection<IEntityReference> articles = entity.getConnections(ARTICLE);
   for (IEntityReference article: articles)
        retVal += ((DoubleValue)article.getAttribute("Weight")).getValue()*
   ((DoubleValue)entity.getAttribute("Amount")).getValue()+
   ((LongValue)entity.getAttribute("Packages")).getValue()*
   ((DoubleValue)entity.getAttribute("PackageWeight")).getValue();
   return new DoubleValue(retVal);
  }
}
```

5. Discussion

Having seen the TME implementation we are going to summarize the most important points in the concept of Transient Model Extensions.

Model-interpretation technique: In contrast to other approaches, which mainly rely on code generation, our approach is based on model interpretation based on a metamodel. [5]. This allows dynamic generation of classes and objects based on a data model.

Single persistency model: TME relies on a single persistent data model, which represents the persistent data structures. This model is used as basis for each extended data model.

Assuming a three layered software architecture with the data source layer holding the persistent data model, TME can be used to create extended data structures in the domain layer and the presentation layer. These data structures are needed for support of additional attributes or relations according to the use cases implemented in each layer.

Separation of concern: TME can be used for implementing a view on the persistent information stored in the application's data source. This is not only important for implementing different software layers. Another use case is the implementation of different components using the same persistent information.

Code modularization: In a method working with TME first the persistent data model is extended and data gathering methods are added, implementing hooks and observers. In a second step the extended model can be applied to an object oriented data cache. So the return value of the method can be based on the extended attributes. Therefore the method can only contain model extension and model querying statements, while calculation can be performed in small code fragments used by the hooks and oberserver. These hooks and observers can be also shared between different model extensions allowing an improved reusability.

5.1. Related persistent data frameworks

To realize the mapping of persistent data in dynamic data structures several widely known technologies exist.

Microsoft's **ADO.NET** framework [12] contains several classes, which enable the usage of relational databases or XML files as persistent data sources. Access to the database is provided through an instance of a DataReader or a DataAdapter. While the first one is only used for reading data from the database, the second allows also data manipulation independent of the database type. The data adapter is used by a dataset component, which is an in-memory database.

The dataset is used by other application layers, but it does not utilizes a data model. Therefore extending the dataset is done at source code level, instead of the model abstraction level.

Hibernate [8] aims at providing an object/relational bridge. It allows the user to work with object oriented concepts like inheritance and composition as well as with database relational concepts such as usage of a DML like SQL. The mapping between the Java classes and the database structure is done by a XML file.

Because Hibernate works at the source code abstraction level, supporting modelbased extensions is not in the scope of this technology.

The **Eclipse Modeling Framework** (EMF) [3] is an implementation related to the OMG Meta Object Framework (MOF). A EMF model is based on a metamodel called ecore model. This model defines the content of eAttributes and eReferences, which belong to the eClass elements in the model.

EMF models can be built from Java files, XML files or UML models stored in XMI. This models can be used as input for Java source code generators, producing class files with annotated source code. This code can be edited manually to add functionality. EMF supports the serialization of objects contained in the EMF model in XMI files.

The ecore model defines attributes allowing to control which elements of the model can be serialized. The transient flag defines whether the corresponding element can be serialized. The volatile flag is used to signal that the value of this element depends on the value of other model elements.

Our approach differs in the following points from the EMF modeling concept:

1. The first aspect is the support of simultanous but independent extensions of the persistency data model by different methods. In our understanding the same persistent data model is used by all extending models, while the extensions are only visible within a particular scope. In contrast, extensions to EMF models are globally visible.

2. The second aspect is the used concept of model extension. While EMF supports extension of classes with subclassing an existent class in a model, our approach directly adds attributes and references to existing classes. The advantage hereby is the constant class type of the extended class, so no modification of code expecting a particular class type is required.

6. Conclusion

This paper introduced a new method to ease the model driven construction of layered data-intensive software. Applying the concepts of data modeling using traditional concepts, results in one data model for each software layer. This leads to redundant class definition in different models with respect to the data source.

In this cases changing the data model of the data source becomes difficult, because all corresponding classes in the other data models need to be changed. Furthermore each model is driven by various requirements, which leads to a different number of attributes in the equivalent class depending on data model.

To overcome this challenge, data models of different software layers can be received applying additional extensions on the data model from the underlying software layer. The mechanism for application of the model changes on the fly is called Transient Model Extension (TME).

This paper presented TME use cases including the extension of classes, class attributes and associations. These transient extensions are driven by layer specific requirements.

To demonstrate the advantages of this method, we presented an example, considering various requirements of the domain layer and the presentation layer. Finally we provided a comparision of our approach to related technologies dealing with persistent data structures.

7. Acknowledgment

We would like to thank our company, Salomon Automation GmbH, for their grant to support our research.

References

[1] Scott W. Ambler. *Agile Database Techniques*. Wiley Publishing, Inc., 2003.

[2] L. Baum and M. Becker. Generic Components to Foster Reuse. In *Proceedings of the 37th International Conference on Technology of Object-Oriented Languages and Systems*, pages 266 – 277, 2000.

[3] Frank Budinsky, David Steinberg, Ed Merks, Raymond Ellersick, and Timothy J. Grose. *"Eclipse Modeling Framework"*. Addison-Wesley, "August" 2003.

[4] Ahmet Demir. Comparison of model-driven architecture and software factories in the context of model-driven development. In *MBD-MOMPES '06: Proceedings of the Fourth Workshop on Model-Based Development of Computer-Based Systems and Third International Workshop on Model-Based Methodologies for Pervasive and Embedded Software (MBD-MOMPES'06)*, pages 75–83, Washington, DC, USA, 2006. IEEE Computer Society.

[5] Ulrich W. Eisenecker and Krzysztof Czarnecki. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, 2000.

[6] Martin Fowler. *Patterns of Enterprise Application Architecture*. Addison Wesly, 2003.

[7] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison Wesley, Reading, MA, 1995.

[8] "Hibernate", "http://www.hibernate.org/5.html". *"Hibernate Reference Documentation"*, "3.2.0ga" edition, 2006.

[9] Stephan J. Mellor, Anthony N. Clark, and Takao Futagami. Model–driven development — guest editor's introduction. *Software, IEEE*, 20(5):14–18, 2003.

[10] OMG. UML Infrastructure, Version 2.0 . Technical Report 2002-09-01, Object Management Group, 2002.

[11] OMG. UML Superstructure, Version 2.0. Technical Report 2002-09-02, Object Management Group, 2002.

[12] David Sceppa. *Microsoft ADO.NET 2.0 Core Reference*. Microsoft Press, 2006.

[13] Gernot Schmoelzer, Stefan Mitterdorfer, Christian Kreiner, Joerg Faschingbauer, Zsolt Kovács, Egon Teiniker, and Reinhold Weiss. The Entity Container — an Object–Oriented and Model–Driven Persistency Cache. In *HICSS'05, Big Island, Hawai'i, USA, Jan. 3-6*, page 277b. IEEE, 2005.

[14] Bran Selic. Model-Driven Development: Its Essence and Opportunities. In *Ninth IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing, 2006. ISORC 2006.*, page 7pp. IEEE, April 2006.

Model-driven Engineering for Development-time QoS Validation of Component-based Software Systems

James H. Hill, Sumant Tambe and Aniruddha Gokhale
Vanderbilt University, Nashville, TN, USA
{hillj,sutambe,gokhale}@dre.vanderbilt.edu

Abstract

Model-driven engineering (MDE) techniques are increasingly being used to address many of the development and operational lifecycle concerns of large-scale component-based systems. One such concern lacking significant research deals with the validation of quality-of-service (QoS) properties of component-based systems throughout their development lifecycle instead of waiting until system integration time, which is very late and can be detrimental to project schedules and costs. This paper describes our novel MDE-based solution to address this challenge. At the core of our solution approach are (1) a set of domain-specific modeling languages that allow us to mimic component "business logic," and (2) a generative programming framework that synthesizes empirical benchmarking code for system emulation and continuous QoS evaluation.

keywords: *model-driven system engineering,continuous QoS validation, code generation.*

1 Introduction

Model-driven engineering (MDE) [27] techniques are increasingly being used to address many of the development and operational lifecycle complexities of large-scale component-based systems. Advances in MDE techniques for component-based systems to date have focused primarily on (a) structural issues of system development, such as component assembly, packaging, configuration and deployment (e.g., the CoSMIC tool chain [7]), and (b) functional and behavioral issues, such as model checking for functional correctness (e.g., Bogor [26] and Cadena [12]) or runtime validation of performance (i.e., running simulations at design time or empirical benchmarks at integration time to validate performance).

Although MDE tools continue to raise the level of abstraction of component-based software systems and address many of their complexities, there remains a major gap in evaluating system quality of service (QoS), e.g., performance and reliability, at different phases of development, which would enable design flaws to be rectified earlier in the development lifecycle. This impediment is due primarily to the "serialized phasing" [27] nature of the development lifecycle wherein the system is developed in layers (e.g., first the components at the infrastructure layer(s) and then the application layer(s)). System QoS validation, therefore, can proceed only when all the system components are available and deployed in the runtime infrastructure. Moreover, waiting too late in the development lifecycle to resolve any performance problems can be too costly to resolve. It is clear that system engineers need proper tools to help address QoS validation not only at integration and production time but at development time before performance problems become too "hard" to locate and resolve.

A promising solution to address the challenge of evaluating QoS at all stages of development entails accurately emulating system components for QoS validation while the "real" components are still being developed. This paper describes our novel MDE-based solution to address the challenges of serialized phasing and QoS validation across the development lifecycle. First, we demonstrate how the problem of serialized phasing can be overcome by emulating component "business logic" using domain-specific modeling languages (DSMLs) [8]. The behavior in our DSML is captured using the formalisms of I/O automata [18] and can be parametrized with executable operations (i.e., workload). Generative programming tools [4] associated with the DSMLs can synthesize empirical benchmarking code for system emulation and QoS evaluation. The QoS evaluation is carried out using the QoS benchmarking framework for component-based systems called the Component Workload Emulator (CoWorkEr) Utilization Test Suite (CUTS) [14].

Paper Organization. The remainder of this paper is organized as follows: Section 2 introduces a case study we use to describe the challenges in realizing a solution for continuous QoS evaluation; Section 3 describes the structure and functionality of our DSMLs for emulating component be-

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

havior and workload; Section 4 explains how we integrated our DSMLs with an existing structural DSML to facilitate code generation for QoS evaluation; Section 5 compares our work with related research; and Section 6 presents concluding remarks.

2 Challenges in Overcoming the Serialized-phasing Barrier

This section describes the challenges in developing a solution that addresses the need for continuous QoS evaluation of component-based systems developed using serialized phasing processes. We use a case study to highlight these challenges.

2.1 A Distributed Stockbroker Application Case Study

We use a representative example drawn from the financial domain [28] as a case study to illustrate the serialized phasing problem and how our research artifacts described in this paper enable us to provide continuous QoS validation [13]. Our case study is called the Distributed Stockbroker Application (DSA), which is an online web application for viewing stock information.

Figure 1 shows a high-level representation of the DSA and its communication flows between components. The DSA is composed of six different components. The *Naming Service* component allows client applications to locate the *Gateway Component* for the application. The location (i.e., the binding IP address and port number) of the naming service component is therefore persistent. The *Gateway Component* serves as the entrance to the stock application, which all clients must pass through. The *Gateway Component* accepts the username and password of the user and sends it to the *Identity Manager* component. The *Identity Manager* component is responsible for verifying the username and password, and initializing the correct QoS policies based on user type. Once the access is granted to the client, it is given direct access to a *Stock Component*. The *Stock Component* is created on-demand and initialized with the correct QoS specified by the *Identity Manager*. The *Stock Component* interacts with a MySQL database that contains the stock information. Lastly, all components in the system – both application and infrastructure – log their activities to a *Logging Component*.

The DSA has two user classes: *Basic* and *Gold*. *Gold* users are persons who use the service frequently, whereas *Basic* users use the service infrequently. Table 1 provides the projected usage pattern and desired response times (i.e., QoS) of each user for the DSA. Due to the serialized-phasing development process, the underlying infrastructure of the DSA (i.e., all the components illustrated in Figure 1) may complete their development at different times. Evaluating system design decisions on the target architecture to locate deployments (i.e., placement of components on hosts) and configurations (i.e., setting of component attributes) that meet the expected QoS therefore has to wait until all the "real" components are available.

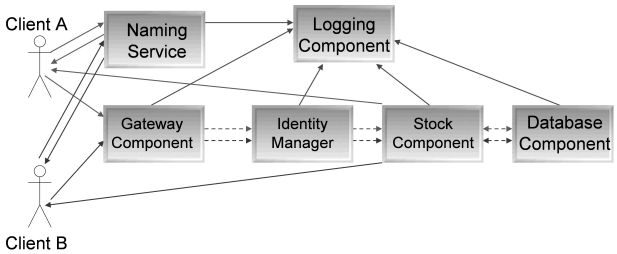


Figure 1. High-level structural composition of the Distributed Stock Application.

Usage Patterns by User Type

Type	Percentage	Response Time (msec)
Basic (Client A)	65%	300
Gold (Client B)	35%	150

Table 1. Predicted usage pattern of the Distributed Stock Application based on user type.

The application components of DSA are implemented as Lightweight CORBA Component Model (CCM) [24] components. The target architecture comprises three hosts for deploying all its components. Lastly, the software platform version is Fedora Core 4 using ACE+TAO+CIAO 5.1 middleware platform available at `www.dre.vanderbilt.edu`.

2.2 Impediments to Overcoming the Serialized-phasing Barrier

To achieve the vision of continuous QoS validation in the presence of serialized phasing, such as in the case of the DSA case study, the proposed solution must address the following challenges:

- **Challenge 1: Emulating business logic** – The emulated components must resemble their the counterparts in both supported interfaces and behavior. Moreover, the emulation environment should allow seamless replacement of faux components with real components as they become available. In the context of the DSA, emulated components should be used to evaluate QoS at early stages of development, and as the "real" components are completed they should replace the emu-

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

lated components to achieve more accurate QoS metrics.

- **Challenge 2: Realistic mapping of emulated behavior** – The behavior specification should operate at a high-level of abstraction (i.e., at the application level) and map to realistic operations (e.g., memory allocations and deallocations, file operations, or database transactions). For example, in the DSA the high-level database behavior should "realistically" query a database for stock information.
- **Challenge 3: Technology independence** – The behavior specification should not be tightly coupled to a programming language, middleware platform, hardware technology, and MDE tool. In the context of the DSA, if we wanted to evaluate the system on CCM or Microsoft .NET [20], or use multiple modeling tools [17,30], then we should be able to reuse the same concepts and models.

The remainder of this paper describes our solution to resolve these challenges.

3 Domain-specific Modeling Languages for Continuous QoS Validation

Addressing the challenges of continuous QoS evaluation in the face of serialized phasing requires mechanisms to mimic application component behavior. This section describes our R&D on two domain-specific modeling languages (DSMLs) [8] called the Component Behavior Modeling Language (CBML) and Workload Modeling Language (CBML). CBML is a DSML for capturing the behavior of a component and WML is a DSML for parameterizing the behavior with "realistic" application-level operations. The remainder of this section discusses both languages in detail explaining how these help resolve Challenges 1 and 2 discussed in Section 2.2 in the context of the case study described in Section 2.1.

3.1 The Component Behavior Modeling Language

Any mechanism that mimics component behavior must incorporate the design principles and semantics of component architectures. In such architectures, systems are composed of components that react to method invocations and events received on their input ports. This "reaction" causes a sequence of activities that can be defined by a series of states and transitions. Although the range of activities performed in the course of a component's execution can vary broadly, they can be divided into two distinct operational classes: *internal* and *communication*.

Internal operations are those not observable from outside a component (e.g., memory allocations/deallocations and database transactions executed by the database component in the DSA case study). Communication operations are representative of sending/receiving an event to/from another component (e.g., input and output events transmitted between each of the components in the DSA case study).

When trying to emulate a component's behavior (i.e., addressing Challenge 1 in Section 2.2), it is desirable to capture it as close as possible to its real counterpart using combinations of internal and communication operations. It is also desirable to represent the behavior based on a formal mathematical foundation because it will (1) facilitate transformation of existing models between different formal behavioral languages (e.g., timed-automata [1], StateCharts [10] and PetriNets [25]), and (2) assist in proving any formal properties of the system (e.g., correctness and stability). Likewise, it will also facilitate reverse transformations (i.e., from models in other languages to models of this language). We believe that lack of formal semantics can limit the capabilities and scope of such a behavior modeling language. At the same time, it should not be dependent on any programming language and software/hardware platform, and be as general purpose as possible.

Based on the desired functionality for modeling component behavior, we have developed the Component Behavior Modeling Language (CBML). CBML is a DSML based on the mathematical formalism of input/output (I/O) automata [18] (details of I/O automata are beyond the scope of this paper). We chose I/O automata as its basis because, analogous to component behavior, I/O automata is ideal for asynchronous and reactive systems. We developed CBML in the Generic Modeling Environment (GME) [17], which is a metamodeling environment that allows the creation of DSMLs and its models. CBML, however, is not coupled to GME since it can be ported to any MDE tool that supports metamodel specification (e.g., Generic Eclipse Modeling System (GEMS) [30]). Developers use CBML to capture component behavior at a high-level of abstraction and use model interpreters to generate configuration and source files for backend emulation and simulation tools. Our current efforts focus primarily on generating source files for emulation tools (see Section 4).

3.1.1 Structure of CBML

As explained in Section 3.1, we developed CBML based on the mathematical formalism called I/O automata [18]. We defined CBML so that it has the necessary subset of elements from I/O automata (illustrated in Figure 2) that will preserve its formal semantics. Users of CBML do not need prior knowledge of I/O automata in order to use CBML.

Figure 3 shows the realization of the CBML artifacts illustrated in Figure 2 using the DSA database component as an example. In CBML, all behavior specification be-

Figure 2. Primary elements for constructing behavior models in CBML.

gins with an *Input Action* element. Each *Input Action* in the behavior model is connected to an initial *State* element. The remainder of the behavior specification is defined by a sequence of *Action* to *State* transitions (similar to I/O automata). For example, the behavior model for the database component in Figure 3 illustrates that an input action causes a query for stock information.

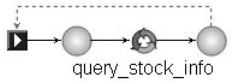


Figure 3. Example CBML behavior model in GME.

To specify the end of a behavior sequence, a *Finish* connection (i.e., the dashed line) is used to connect the final *State* to the starting *Input Action*. We require this connection because we allow sharing of behavior sequences to simplify modeling (illustrated in Figure 4). For example, the DSA has two type of users who have the same behavior. It is possible to model each person's input to the database component (or any component) separately but share the same behavior as illustrated in Figure 4. The explicit finish connections therefore help resolve ambiguity when determining where each user type's behavior terminates.

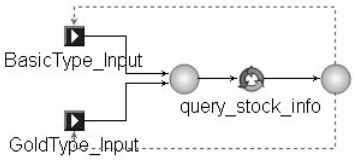


Figure 4. Example of sharing behavior in CBML.

Specifying output actions in CBML

I/O automata defines behavior as an input action that causes a series of "internal" operations and results in a set of output actions, if any. To be consistent with I/O automata semantics, we allow behavior specifications to include output operations, however, output actions have the same *modeling* semantics as I/O automata internal operations (e.g., CBML *Action* element). We made this design choice because, similar to CBML action elements, output actions can also have a series of action-to-state transitions after completing a single output action.

Figure 5 illustrates an example behavior model with output actions, which are represented by the two rightmost squares with the triangle, for the database component in the DSA. After the component completes its query to the database for stock information, it sends the information back to the requester, and sends a status message to the logging component.

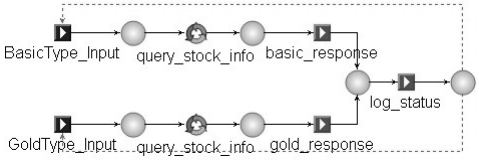


Figure 5. Example CBML behavior model with output actions.

Preconditions, postconditions, and variables in CBML

CBML allows a user to define variables in behavior models to stay consistent with the I/O automata semantics. The purpose of a variable is to preserve information that represents the current state of the system or component. As illustrated in Figure 6, a variable is represented by the element with the star image. Users use variables in their behavior model by referencing them in the preconditions and postconditions of the transition (i.e., connection from a state to an action), and effect (i.e., connection from an action to a state) connections, respectively. This allows developers to create more "realistic" behavior models, such as counting the number of users of each type executing queries on the database and/or guarding a workload until the system reaches a certain state.

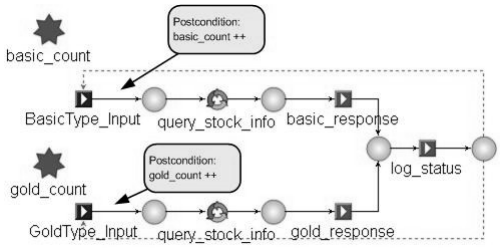


Figure 6. Example CBML behavior model with variables.

Domain-specific extensions in CBML

Some input events that are critical in the domain of component-based systems (e.g., lifecycle events such as *activation* and *passivation* or monitoring notification events such as degradation of QoS) are not first class entities in I/O

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

automata. I/O automata does not distinguish between these kinds of events because it is a general-purpose language that is not tied to any particular domain (e.g., component-based systems). We therefore extended I/O automata (without affecting its formal semantics) in CBML to capture this aspect of component behavior more expressively as discussed below and illustrated in Figure 7:

- **Environment Events –** represent input actions to a component that are triggered by the hosting system rather than another component (e.g., lifecycle events from the hosting container or fault-tolerance notifications to serialize the state of a component).
- **Periodic Events –** represent input actions from the hosting environment that occur periodically (e.g., setting/receiving a timeout event to periodically transmit status updates). We also allow a probability to be associated with periodic events to provide non-deterministic behavior.

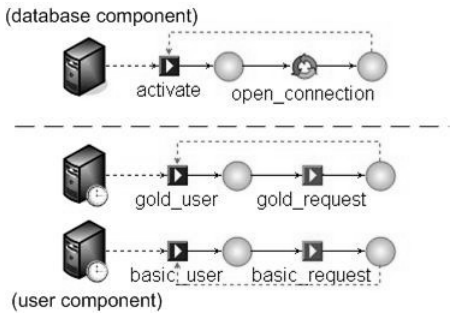


Figure 7. CBML's domain-specific extensions to I/O automata

In the context of the DSA, when the database component is activated it creates an initial connection to the database (illustrated in Figure 7). Likewise, we can use periodic events to model the behavior of each user type by associating each one with correct probability (e.g., 0.35 and 0.65 for Gold and Basic type, respectively) and sequencing it with an output event within a "user" component (also illustrated in Figure 7).

Usability extensions in CBML

One of the main goals of defining behavior at a high-level of abstraction is simplicity and ease of use. If the size of the behavior model is "huge" and CBML adheres strictly to its current representation of I/O automata, its ease of use is compromised because one of the major drawbacks of many automata languages is scalability [10]. To address this issue we defined the following usability extensions, which do not affect the underlying I/O automata semantics:

- **Composite Action –** is a modeling element that contains other actions. It allows developers to create groups of action-to-state sequences that can help reduce the amount of clutter in the model. A composite action has the same modeling semantics as a regular action, however, we defined a constraint that requires composite actions to contain only a single input action. This is necessary because composite actions encapsulate a single, reusable behavior workflow, and not multiple behavior workflows.
- **Log Action –** is an attribute of an *Action* element that determines if the action should be logged. The semantics of "logged" are dependent on how the model is interpreted. For example, a modeler might choose to log "network send" actions and not "memory allocation" actions.
- **Repetitions –** is an attribute of an *Action* element that specifies how many times to repeat the operation. This prevents the same action from having to be specified multiple times in order to achieve repetition. It is clear that setting this value to zero implies the action is disabled. This allows developers to seamlessly bypass an action temporarily without actually removing it.

To address the usability concerns in the modeling aspect, we also developed a GME add-on that assists users in creating models rapidly by auto-generating required elements (e.g., states) and connections depending upon the context. Although this feature is GME-specific, most MDE tools provide support for implementing features that help improve user experience [29].

3.2 The Workload Modeling Language

The Component Behavior Modeling Language (CBML) described in Section 3.1 gives developers the ability to model behavior via generic actions and properties. For analysis techniques, such as simulation, CBML is enough to capture the behavior of the component (e.g., its actions, states, and respective transitions), which can be interpreted to define configuration files for simulation tools. For emulation purposes, however, these actions do not exemplify the "business logic" of components because it does not capture the workload of reusable objects within a component (e.g., objects and their methods). Moreover, when defining the workload of components using CBML, it is "hard" to specify realistic workloads that map to executable operations for an emulated component. To address this challenge (i.e., Challenge 2 in Section 2.2) we developed the Workload Modeling Language (WML).

WML is a middleware and hardware platform-independent and programming language-independent DSML that allows developers to define workload generators that contains actions to represent realistic operations

(e.g., memory allocations/deallocations and database transactions) at a high-level of abstraction. Model interpreters associated with WML parse the constructed models and use generative programming techniques to map the abstract representation to executable operations in the target programming language and platform (see Section 4).

We implemented WML in GME, but similar to CBML, it can be ported to any MDE tool that supports metamodel specification. The remainder of this section discusses WML in detail.

3.2.1 Structure of WML

WML is a DSML that allows developers to create workload generators (called "workers") with executable actions for emulation. Figure 8 illustrates the compositional overview of WML. We designed WML using a hierarchical structure that resembles common object-oriented programming packaging techniques to be consistent with conventional component technologies.

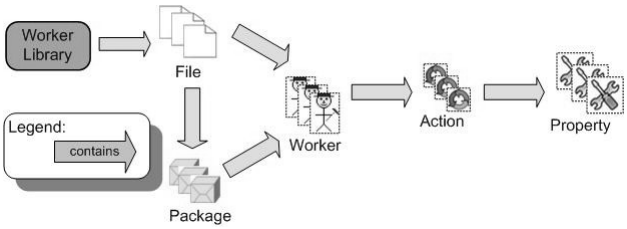


Figure 8. High-level compositional overview of WML.

The outermost containment elements in WML are *library* elements. Library elements represent reusable containers (e.g., modules) for grouping common *workers*. The library elements are composed of multiple *files*, which represent a concrete location on disk that defines its contained workers (recall that workers are workload generators). File elements can contain *packaging* elements that act as a scoping mechanism so that *workers* can have the same name and appear in the same file (similar to C++ namespaces or Java packages). *Workers* contain executable actions that represent its "business logic" operations (or workloads). Lastly, actions can contain optional *properties* that define configurable parameters (e.g., input arguments) for the action executed by the parent.

3.2.2 Parameterizing CBML with executable operations

When WML is integrated with CBML, it enables developers to model the component behavior using executable operations. From a modeling perspective, *workers* in WML have the same modeling semantics as *variables* in CBML, and worker *actions* in WML have the same modeling semantics as *actions* in CBML. This design feature allows us to integrate WML with CBML.

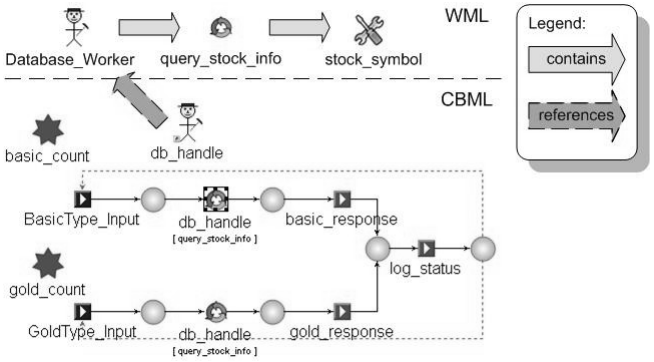


Figure 9. Example CBML model parameterized with WML actions.

Figure 9 illustrates the behavior model of the database component from Figure 6 in Section 3.1.1 that has been parameterized with WML actions. The top portion of the image illustrates the WML composition for a database worker. In the bottom portion of the image, the actor (i.e., `db_handle`) is a *variable* that references the database worker. The *action* is a modeling instance of the worker action in the top portion of the image whose name must match the name of its parent worker variable. We made this design requirement because it (1) helps resolve ambiguity when determining what action belongs to what parent since it is possible to include the same worker variable type multiple times in a behavior model, and (2) reduces modeling clutter as opposed to explicitly creating a directed connection between a parent and its action.

4 Technology Independent Approach to Continuous QoS Validation

In Section 3 we described two behavioral DSMLs: CBML and WML that illustrated how integrating both languages allowed us to emulate the behavior (Challenge 1 of Section 2.2) and mapping the behavior to realistic operations (Challenge 2 of Section 2.2). Although WML allowed us to parameterize the generic actions in CBML with executable operations, these models are insufficient to generate emulation code directly without knowing the structural composition of the system and its components since the latter determines the end-to-end workflows whose QoS validation is more interesting and important to system developers.

We therefore integrated CBML and WML with the Platform Independent Component Model Language

(PICML) [2] because the latter captures the structural aspects of a system and its components, which is necessary when generating source code for components that resemble the real counterparts. Moreover, since both PICML and CBML/WML provide platform and programming language independent modeling capabilities, their integration and model interpretations provide a technology independent approach to continuous QoS evaluation (Challenge 3 in Section 2.2).

Although we chose PICML as the structural DSML to integrate CBML and WML, the concepts presented in Section 4.1 can be applied to any structural DSML provided that it clearly differentiates between input and output ports of a component. The remainder of this section discusses integration of CBML and WML with existing languages (e.g., PICML in CoSMIC), and how our approach to generating emulated logic for components that mimics their real capabilities is decoupled from the underlying platform and programming language technology.

4.1 Integrating Behavioral and Structural DSMLs

Domain-specific modeling languages (DSMLs), such as PICML, allow developers to model different ports of a component (e.g., facet/receptacles and event sources/sinks). The facets/event sinks represent inputs to a component, while receptacles/event sources represent outputs from a component. Structural DSMLs, however, capture structural input/-output (I/O) elements without any correlating behavior (i.e., there is no clean representation to associate the I/O elements of structural models with the I/O actions in behavioral models). We therefore defined a set of "connector elements" that enable developers to connect the I/O elements in the structural model with their corresponding I/O elements in the behavioral model.

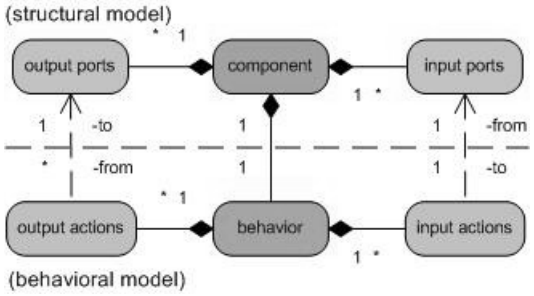


Figure 10. Conceptual model of integrating behavioral and structural DSMLs.

Figure 10 illustrates how structural DSMLs (e.g., PICML) which define components that have I/O ports and behavioral DSMLs (e.g., CBML and WML) that have I/O actions can be integrated by having the structural DSML "contain" the behavioral DSML. In particular, we require a component to contain the behavior. Additionally, we define a modeling connection between the input port and input action, but require that the name of the output action match the name of the corresponding output port. We made this design decision because explicitly defining a connection between an output action and port will clutter the model since there is a many-to-one mapping between an output action and an output port.

To further illustrate this concept, Figure 11 shows how the integration of the DSMLs is realized. The outer rectangle of Figure 11 illustrates the PICML model of the database component. The inner rectangle highlights the same database component with CBML and WML from Figure 9 integrated into PICML, thus allowing us to model the same behavior exemplified in Section 3 with its respective structure (e.g., interface and attributes).

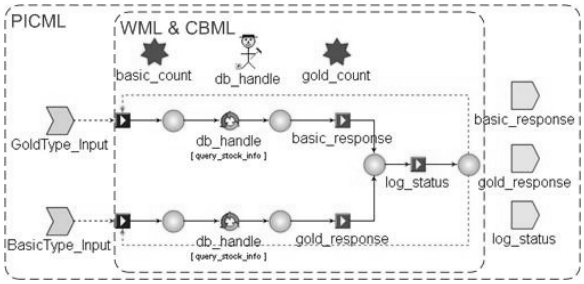


Figure 11. Realization of integrating CBML and WML with PICML in CoSMIC.

4.2 Code Generation for Emulation

This section describes our approach for achieving code generation for emulation, which enables us to conduct continuous QoS validation during system development. Our current effort allows developers to generate emulation code for the *Component Workload Emulator (CoWorkEr) Utilization Test Suite (CUTS)*, however, our code generation architecture is not dependent on CUTS (e.g., Challenge 3 in Section 2.2). Figure 12 illustrates a conceptual model of our code generation architecture, which is composed from three technology independent, but language dependent layers of abstraction:

- **Emulation** - This layer represents the application layer's "business logic". The elements in WML used to parameterize the CBML behavior are mapped to this layer when model interpreters parse the model. For example, the *query_stock_info* action is generated at this layer in C++ code.
- **Templates** - This layer acts as a bridge [6] between the upper emulation layer and lower benchmarking layer. This allows both layers to evolve independently of

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

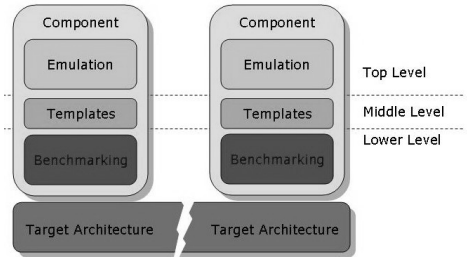


Figure 12. Code generation architecture for emulation.

```
void DatabaseComponent::push_BasicType_Input (
  QueryEvent * ev ACE_ENV_ARG_DECL_WITH_DEFAULTS)
  ACE_THROW_SPEC ((::CORBA::SystemException))
{
  // get activation record for this thread
  CUTS_Activation_Record * record = CUTS_THR_ACTIVATION_RECORD ();
  this->basic_count_ ++;

  record->perform_action_no_logging (
    CUTS_Database_Worker::query_stock_info (this->db_handle__));

  CUTS_CCM_Event_T <OBV_QueryResponse> __event_100000028__;
  this->context_->push_basic_response (__event_100000028__.in ());

  CUTS_CCM_Event_T <OBV_LogStatus> __event_100000029__;
  this->context_->push_log_status (__event_100000029__.in ());
}
```

Figure 13. Excerpt of generated code from a PICML model extended with CBML and WML.

each other. For example, if we want to provide support for other benchmarking frameworks we do not have to alter the generated code because the templates will provide the mapping. Likewise, if we ported the DSA to a different technology (or language) the code generator can tailor the source code to plug into this layer given we support the target programming language.

- **Benchmarking** - This layer represents the underlying benchmarking framework (e.g., CUTS). Methods in this layer are invoked by the template layer above to capture workload metrics, such as execution timing of a database query by the database component, or response time of each user type in the context of the DSA.

Lastly, the encapsulating object for each of the three layers is the actual *component* hosted by the *target architecture*, which is language and technology dependent. The component is generated so that is has the same structure as its "real" counterpart (e.g., same interfaces and attributes). Figure 13 illustrates the generated code for a portion of the database component in the DSA. As illustrated in Figure 13, the *push_BasicType_Input* method is the realization of implementing the *BasicType_Input* input event port in CCM. Each line of source code represents the WML actions used to parameterize the CBML behavior. The *record* is the template object that allows the emulation operations to be adapted for monitoring and analysis by the CUTS benchmarking framework.

Generation for Simulation. Although our efforts currently focus on generating emulation code, it is possible to create model interpreters that generate configuration files for simulation tools such as UPPAAL [16]. This will allow developers to utilize the features of a simulation tool such as validating correctness, evaluating preconditions and postconditions, and checking for reachability [15]. Moreover, it will alleviate the time and effort required to manually produce these configuration files, which can be error prone and tedious.

5 Related Work

Statecharts [10] gained widespread usage when they were integrated with the STATEMATE [11] modeling tool, and since then a variant has became part of UML (i.e., UML Statecharts) [5]. Similar to CBML, statecharts can be used to describe behavior of large complex systems. CBML extends Statecharts by clearly separating component behavior from workloads using WML. The generative techniques associated with variants of statecharts are targeted towards simulation and runtime verification [15,21]. Our generative techniques can be extended to simulation and runtime verification tools [16] as well, however, it extends UML statecharts efforts [22] because it facilitates seamless replacement of the emulated components (i.e., components generated from behavior models) with the real components as they become available. Furthermore, our generative techniques and concepts are not tied to a specific technology or tool, whereas the technique presented in [22] et al., is bound to a specific tool.

The Abstract State Machine Language (AsmL) [9] developed at Microsoft Research is an executable specification language based on the theory of Abstract State Machines. AsmL is useful when developers need precise, non-ambiguous methods to specify a system, either software or hardware. AsmL, however, is not a graphical modeling language like CBML and WML. Furthermore, users of CBML and WML operate at a high-level and do not require in-depth knowledge of the underlying formalism, whereas AsmL requires developers to have some understanding of abstract state machines and programming formalisms, which can restrict its applicability (e.g., for system testers who have no knowledge of complex formalisms or programming).

Executable UML (xUML) [19] and the Action Language [23] are both for defining workload that can map to the desired target architecture. WML is orthogonal to both

xUML and the Action Language efforts, however, WML operates at higher level of abstraction. xUML and the Action Language require developers write abstract implementation code, which requires knowledge of programming semantics, whereas WML leverages pre-existing objects and methods (i.e. workload generators) that are not defined by the user for code generation.

WinFX Workflow [3] is a modeling language developed by Microsoft et al., which is a part of the Windows Workflow Foundation. Similar to CBML, WinFX allows developers to express workflows but it is coupled with workload. WinFX also facilitates code generation, but is confined to the Microsoft .NET framework whereas our generative programming technique is technology and tool independent and can be applied to multiple middleware platforms including Microsoft .NET.

6 Concluding Remarks

This paper described a model-driven generative programming approach to address the challenges of evaluating component-based system QoS throughout the development lifecycle instead of delaying it to integration time. Our approach defined two modeling languages, namely CBML and WML, that capture the behavior of application components at a high-level. We then integrated these DSMLs with PICML, which models structural properties of applications. Lastly, we used model interpreters to map the behavior specifications to executable operations that leverage existing emulation frameworks, such as CUTS.

This approach allows for continuous integration and QoS validation of the system because as more is learned about the components, the behavior can be refined and regenerated for emulation. Likewise, as the real application components are ready, they can replace the emulated components and their impact on system QoS can be observed. We expect the results of real versus emulated components to match provided the behavioral models of the emulated components approximate the real component behavior closely.

6.1 Lessons Learned

Model-driven Engineering comprising the use of DSMLs and generative programming provides an effective solution to address the challenges facing development lifecycles of next generation, large-scale software systems. Several challenges were encountered during the development of CBML and WML and several challenges remain to be resolved. Our experience developing and using the MDE framework described in this paper suggests the following benefits:

- Using a DSML based on a mathematical formalism to define behavior of components helps in specifying unambiguous behavior when generating code and configuration files for emulation and simulation.
- Separating the workload, behavior, and structural models allows all to evolve independently of each other. Moreover, it encourages the same behavior model to be supported in multiple structural models to increase portability, flexibility, and usability.
- Using generative programming with templates that are parameterized by actions from behavioral models allows the DSML to easily be adapted to different environments of execution (e.g, benchmarking environments or real-world deployment).

6.2 Future Work

Although our approach of integrating a behavior and workload modeling language with a structural language has many benefits and addresses many challenges of the "serialized-phasing" process, there is also room for improvement and future work:

- Despite the ability to capture behavior of a component and its state, data flow of a component can only be defined based on state variables. In real world, properties of input actions (e.g., event values) can affect the flow of execution in a real component. We therefore need to extend CBML with a simple programming language that will allow developers to use such properties when defining behavior.
- As the "real" components become available and replace the emulated components, it is ideal to capture workload metrics of the real component. We therefore need to extend the current capabilities of both modeling languages and code generators to handle evaluation of real components (i.e., benchmarking them using "realistic" input data).
- The workload generators (i.e., *workers*) in WML resemble class objects in object-oriented programming languages. We therefore need to extend the capabilities of WML such that existing class objects in the target programming language can be used in WML models. Moreover, these extensions will enable "real" implementation code to be generated directly from models.

By providing these extensions to our MDE approach, we will be able to continue addressing many of the challenges component-based system developers experience when they face time-to-market and product quality pressures.

References

[1] R. Alur and D. L. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[2] K. Balasubramanian, J. Balasubramanian, J. Parsons, A. Gokhale, and D. C. Schmidt. A Platform-Independent Component Modeling Language for Distributed Real-time and Embedded Systems. In *Proceedings of the 11th Real-time Technology and Application Symposium (RTAS '05)*, pages 190–199, San Francisco, CA, Mar. 2005. IEEE.

[3] D. Box and D. Shukla. WinFX Workflow: Simplify Development with the Declarative Model of Windows Workflow Foundation. *MSDN Magazine*, 21:54–62, 2006.

[4] K. Czarnecki and U. W. Eisenecker. *Generative Programming: Methods, Tools, and Applications*. Addison-Wesley, Reading, Massachusetts, 2000.

[5] B. P. Douglass. UML Statecharts. `www-md.e-technik.uni-rostock.de/ma/gol/ilogix/umlsct.pdf`.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Reading, MA, 1995.

[7] A. Gokhale, K. Balasubramanian, J. Balasubramanian, A. S. Krishna, G. T. Edwards, G. Deng, E. Turkay, J. Parsons, and D. C. Schmidt. Model Driven Middleware: A New Paradigm for Deploying and Provisioning Distributed Real-time and Embedded Applications. *The Journal of Science of Computer Programming: Special Issue on Model Driven Architecture*, 2005.

[8] J. Gray, J. Tolvanen, S. Kelly, A. Gokhale, S. Neema, and J. Sprinkle. Domain-Specific Modeling. In *CRC Handbook on Dynamic System Modeling, (Paul Fishwick, ed.)*. CRC Press, Dec. 2006.

[9] Y. Gurevich, B. Rossman, and W. Schulte. Semantic Essence of AsmL. *Theoretical Computer Science*, 343(3):370–412, 2005.

[10] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.

[11] D. Harel, H. Lachover, A. Naamad, A. Pnueli, M. Politi, R. Sherman, A. Shtull-Trauring, and M. B. Trakhtenbrot. STATEMATE: A Working Environment for the Development of Complex Reactive Systems. *Software Engineering*, 16(4):403–414, 1990.

[12] J. Hatcliff, W. Deng, M. Dwyer, G. Jung, and V. Prasad. Cadena: An Integrated Development, Analysis, and Verification Environment for Component-based Systems. In *Proceedings of the 25th International Conference on Software Engineering*, Portland, OR, May 2003.

[13] J. H. Hill and A. Gokhale. Continuous QoS Provisioning of Large-scale Component-based Systems using Model Driven Engineering. In *Proceeding of ACM/IEEE* 9th *International Conference on Model Driven Engineering Languages and Systems (MoDELS) (poster session)*, Genova, Italy, October 2006.

[14] J. H. Hill, J. Slaby, S. Baker, and D. C. Schmidt. Applying System Execution Modeling Tools to Evaluate Enterprise Distributed Real-time and Embedded System QoS. In *Proceedings of the 12th International Conference on Embedded and Real-Time Computing Systems and Applications*, Sydney, Australia, Aug. 2006. IEEE.

[15] D. Huang and H. Sarjoughian. Software and Simulation Modeling for Real-Time Software-Intensive Systems. In *Proceedings of the Eighth IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'04)*, pages 196–203, Washington, DC, USA, 2004. IEEE Computer Society.

[16] K. G. Larsen, M. Mikucionis, and B. Nielsen. Online testing of real-time systems using uppaal. In *FATES*, pages 79–94, 2004.

[17] A. Ledeczi, A. Bakay, M. Maroti, P. Volgysei, G. Nordstrom, J. Sprinkle, and G. Karsai. Composing Domain-Specific Design Environments. *IEEE Computer*, pages 44–51, November 2001.

[18] N. Lynch and M. Tuttle. An Introduction to Input/Output Automata. *CWI-Quarterly*, 2(3):219–246, September 1989.

[19] S. J. Mellor, M. J. Balcer, S. Mellor, and M. Balcer. *Executable UML: A Foundation for Model Driven Architecture*. Addison-Wesley Professional, May 2002.

[20] Microsoft Corporation. Microsoft .NET Development. `msdn.microsoft.com/net/`, 2002.

[21] M. Naughton, J. McGrath, and D. Heffernan. Real-time Software Modelling using Statecharts and Timed Automata Approaches. In *Proceedings of the IEE Irish Signals and Systems Conference*, Dublin, Ireland, June 2006.

[22] I. A. Niaz. Code Generation From Uml Statecharts.

[23] S. Nordstrom, S. Shetty, D. Yao, S. Ahuja, S. Neema, and T. Bapty. The Action Language: Refining a Behavioral Modeling Language. In *Proceedings of the 12th IEEE International Conference on the Engineering of Computer-Based Systems (ECBS 2005)*, Piscataway, NJ, USA, 2005. IEEE.

[24] Object Management Group. *Light Weight CORBA Component Model Revised Submission*, OMG Document realtime/03-05-05 edition, May 2003.

[25] J. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice Hall, Englewood Cliffs, NJ, 1981.

[26] Robby, M. Dwyer, and J. Hatcliff. Bogor: An Extensible and Highly-Modular Model Checking Framework. In *Proceedings of the* 4th *Joint Meeting of the European Software Engineering Conference and ACM SIGSOFT Symposium on the Foundations of Software Engineering (ESEC/FSE 2003)*, Helsinki, Finland, September 2003. ACM.

[27] D. C. Schmidt. Model-Driven Engineering. *IEEE Computer*, 39(2):41–47, 2006.

[28] T. Taibi, L. B. Ping, N. S. Wen, L. K. Sing, and C. K. Lim. Developing a Distributed Stock Exchange Application using CORBA. In *Proceeding of the Student Conference on Research and Development (SCOReD)*, Putraiaya, Malaysia, 2003.

[29] B. Trask and A. Roman. Model Driven Engineering Basics using Eclipse. In *Proceeding of ACM/IEEE* 9th *International Conference on Model Driven Engineering Languages and Systems (MoDELS)*, Genova, Italy, October 2006.

[30] J. White and D. C. Schmidt. Simplifying the Development of Product-line Customization Tools via Model Driven Development. In *Proceedings of the MODELS 2005 workshop on MDD for Software Product-lines*, Half Moon Bay, Jamaica, Oct. 2005.

A Partitioning Analysis of the .NET Common Language Runtime

Joshua R. Dick
Faculty of Computer Science
University of New Brunswick
Fredericton, New Brunswick, Canada
y0h7s@unb.ca

Kenneth B. Kent
Faculty of Computer Science
University of New Brunswick
Fredericton, New Brunswick, Canada
ken@unb.ca

Joseph C. Libby
Faculty of Computer Science
University of New Brunswick
Fredericton, New Brunswick, Canada
g6x2d@unb.ca

Abstract

Microsoft's .NET platform was developed to simplify development of Windows applications. At the core of the .Net platform is a virtual machine known as the Common Language Runtime (CLR). Virtual machines do not allow for optimal performance, and a full hardware implementation is not always feasible. The goal of this paper is to present a preliminary partitioning scheme upon which future refinements can be made, and to analyze the performance of the partitioning scheme based on instructions executed in each partition and the data to be passed between partitions. Conclusions and recommendations as to implementation of the hardware partition are given to aid in future implementations of a hardware/software co-designed CLR.

1 Introduction

Microsoft's .NET platform was developed to simplify development of Windows applications by providing a robust framework through which a developer can manage applications and interact easily with the operating system. At the heart of this platform is an execution environment known as the Common Language Runtime (CLR) [1]. In Microsoft Windows, CLR is implemented as a Just-In-Time compiler, thus a .NET application does not interact directly with the host computer. This extra level of abstraction results in slower execution due to the JIT compiling each method to native code the first time it is executed. A full hardware implementation would be able to overcome the overhead created by the JIT; however a full hardware implementation is not always feasible. Thus a hardware/software partitioning of the CLR should be considered to get some of the performance gains of a hardware implementation while overcoming some of the difficulties of fully implementing the CLR into hardware. Similar research has been performed using the Java Virtual Machine in [2] [3] [4].

This paper presents a partitioning scheme based on the description of the instructions given in the ECMA 355 standard [5] and the data collected from analyzing the CLR as presented in [1]. This partitioning scheme is a preliminary one upon which future refinements may be made in order to maximize performance or other design constraints. For the purpose of this paper benchmarking results were gathered using Southern Storm Software's PNetMark .Net benchmarking package.[6] The benchmark is executed with the partitioning in place by means of tagging instructions that are to be executed in hardware and instructions that are to be executed in software and gathering relevant local variable, constant pool, and stack data accordingly.

2 Partitioning Scheme

When considering an initial partitioning scheme for a virtual machine it should be considered how well the low level instruction set maps to typical CPU instructions. The low level instructions are well suited for hardware implementation because such instructions are typically used in modern microprocessors and advanced techniques already exist for their execution, such as pipelining and parallel execution. The high level instructions are well suited for the software partition due to their need to access the computer system's components, their dynamic memory usage and their virtual machine management functionality.

For these reasons it was decided that the initial partitioning explored here is based on these principles. Partition III of the ECMA specification outlines the CLR instructions in two categories: Base Instructions and Object Model Instructions. Base Instructions are described as being *Turing Complete*, meaning that they perform all operations put forth by Alan Turning to define a computationally complete machine, and are closely related to those instructions that are typically found in modern microprocessors [7]. *Object Model* instructions are instructions that provide object-oriented programming support and high level operating system support to the virtual ma-

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

chine. Thus these categories are used as a basis for the initial partition explored here.

Following this partitioning scheme, the partitions can be detailed as follows. The software partition encompasses three main categories of instructions: *Scalar Data Transfer*; *Control Flow*; and *Object Manipulation*. These operations all require high level memory checking and loading when executed. The operations in the software partition that are executed in the benchmark are outlined in Table 1.

Category	Subcategory	Sub-subcategory	Instructions
Scalar Data Transfer	Metadata Accessing		ldsfld
			stsfld
Control Flow	Subroutine Call		call
Object Manipulation	Array Specific		ldlen
			ldelem.i4
			stelem.i4
	Object-Specific	Field Accessing	ldfld
			stfld
		Object Creation	newarr
			newobj
			ldstr
			box
		Method Handling	callvirt

Table 1: Executed Instructions in the Software Partition and their Categories

- *The Scalar Data Transfer* operations are those that access Metadata, i.e. class data that may or may not exist in memory when executed.
- *The Control Flow* operations are only those instructions that call subroutines. The instruction may call a subroutine that does not exist in memory yet, and may thus require high level checking and loading.
- *Object Manipulation* operations are those that access arrays, objects and methods. They are also responsible for method calling. These operations may all require data that is not loaded into memory, and again may require high level checking and loading.

The hardware partition encompasses many more instructions than the software partition, covering four main instruction categories: *Scalar Data Transfer*; *Stack*; *ALU*; and *Control Flow*. As mentioned earlier, these instructions are similar to typical CPU operations and are well suited for hardware implementation. These instructions are outlined in Table 1 and are explained as follows:

- *Scalar Data Transfer* operations are those that load and store local variables.
- *Stack* operations are the operations that push constants onto the stack, push immediate values onto the stack, perform duplication of the variable on the top of the stack, and pop the variable off the top of the stack.
- *Arithmetic* operations are the instructions that perform mathematical, logic and conversion operations.
- *Control Flow* operations consist of operations that change the program counter, either by a conditional instruction, an unconditional instruction, or return from a subroutine.

3 Performance of Partitions

Once a partitioning has been decided upon it must be determined whether the partitioning is suitable for hardware implementation. When considering an instruction set partitioning, such as the one proposed here, the analysis can be broken down into three basic areas: instruction execution location; context switching; and the level of data exchanged between partitions. These areas are explained and their data is examined in the following sections.

3.1 Instructions Executed in the Software Partition

The first step in evaluating the software partition is to analyze how many instructions are executed in it. Of all the instructions executed in the benchmark, 13.33% are executed in the software partition which indicates good coverage by the hardware partition.

It is important to consider how many instructions are executed consecutively when execution is taking place in a partition. A small amount of instructions executed sequentially in a partition is indicative of many unnecessary context switches. Figure 1 shows that only one or two instructions are executed in the software partition before a context switch back to hardware, with a mean of 1.07 and a standard deviation of 0.34. Of these, over 93% are only executing one instruction.

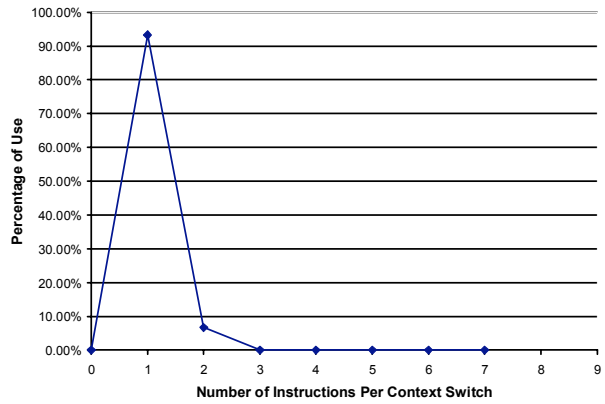


Figure 1: Instructions Executed in Software Partition

3.2 Instructions Executed in Hardware Partition

The hardware partition is responsible for executing 86.67% of all instructions. This indicates that the majority

of instructions are executed in the faster hardware partition, which is an indicator of good partitioning. Figure 2 outlines how many instructions are executed consecutively in the hardware partition. A greater spread of the number of instructions executed consecutively exists in this partition than in the software partition. Most (over 95%) switches to hardware execute between one and fifteen instructions sequentially, the remaining 5% spread out relatively evenly up to 813 instructions in a row. The mean is 7.04 instructions with a standard deviation of 146.23. This indicates that the hardware partition performs well in most cases.

Switching contexts is a time consuming operation and when only one or two instructions are executed in a partition, the context switch is not worth the tradeoff of executing the instructions in hardware. To overcome this, a module that reads future instructions should be implemented. Such a module would be able to determine how many of the future instructions will cause context switches to hardware, and determine when such context switches are appropriate.

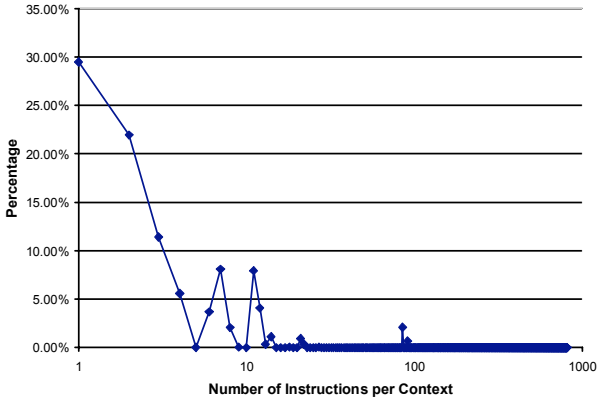


Figure 2: Instructions Executed in Hardware Partition

4 Context Switching

Context switching refers to the program transferring execution between the hardware and software partitions. For example, if a software instruction is executing and the next instruction is a hardware instruction a context switch will occur. When a context switch occurs, the current state of the virtual machine must be passed to the target partition. This includes the method data, class data, the stack, and the method bytecode. Thus switching between partitions is a time consuming operation, and thus can degrade performance of a co-designed system if they occur unnecessarily. Context switching should be considered when partitioning a system.

4.1 Instructions Generating Context Switches

Figure 3 outlines the instructions that cause context switches from the software partition to the hardware partition where 29 of the 46 hardware instructions cause a context switch. Loading variables, constants and arguments cause context switches more frequently than others. The rest of the instructions cause context switches in similar proportions. This indicates that pre-loading all required variables, arguments and constants into the hardware partition prior to execution would be beneficial to execution.

Figure 4 outlines the instructions that cause context switches from hardware where 11 of the 13 software instructions cause context switches. *call, ldfld, and ldelem.i4* cause most of the context switches; they cause nearly 92% of all context switches from hardware to software. The remaining instructions cause very few context switches.

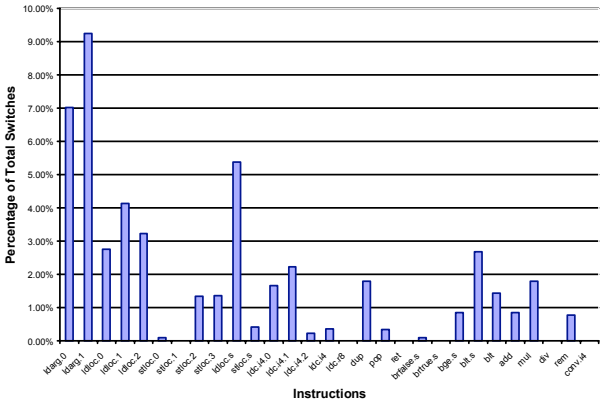


Figure 3: Instructions that result in Context Switches to Hardware

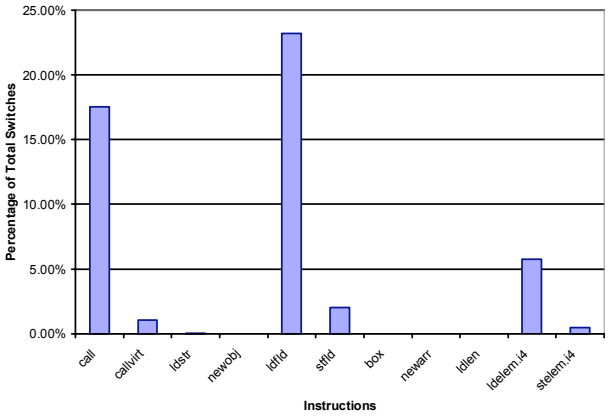


Figure 4: Instructions that result in Context Switches to Software

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

5 Data Movement between Partitions

Another measure in the performance of a partitioning scheme is to monitor the amount of data passed between partitions on context switches. This is because it requires time to pass that data to hardware, which has an impact on performance. Ideally the amount of data to be passed would be zero, however this is obviously impossible. In order to measure the data we need to consider what data is passed for execution. Each method has five essential data areas that are required for execution: program counter; local variables; constant pool; stack; and method size. The program counter associated with each method is used to monitor the next instruction to be executed, thus this register must be passed between the hardware and software partitions. Local variables may be required by each method to store and access values during execution, thus this data must be passed between partitions. The constant pool is class data that may be required by each method to, again, store and access data during execution. Thus these values must be passed between partitions during execution. The stack is used for intermediate values by instructions during execution. Finally, the method size refers to the number of bytecodes required by the method to execute. These bytecodes must be passed to the hardware partition in order for the method to run. This data is vital and must be passed between the hardware and software partitions. The details of this data are outlined in the following sections.

5.1 Local Variables

Figure 5 outlines the number of local variables passed between partitions, either to hardware from software or to software from hardware. Since the number of switches to hardware is equal to the number of switches to software and the number of local variables in a method does not change, only one graph is required to detail the data movement between partitions. Nearly all methods in the benchmark that cause context switches use local variables, only slightly more than 0.0002% of all context switches need not pass any local variables. Between zero and fourteen local variables are passed between partitions, with an average of 5.61 and standard deviation of 3.99. Most, just over 65% of context switches, require passing four, six or ten local variables. The remaining 35% pass zero, one, two, three, and fourteen local variables.

Passing local variables to the hardware partition does not present a major problem. A modest system bus width of 64-bits running at 66 MHz would be sufficient to load the maximum number of local variables very quickly. Assuming the local variables have a size of 32-bits, 14 could be loaded in seven clock cycles.

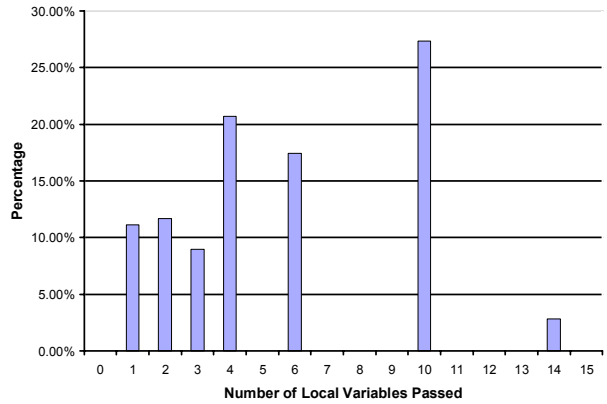


Figure 5: Number of Local Variables Passed Between Partitions

5.2 Constant Pool

The amount of constant data passed between partitions is outlined in Figure 6. As with local variables, the amount of constant data does not change through execution and only one graph is needed to describe the amount of data passed to either partition. The amount of data passed is largely dependant on the classes used and how many context switches occur in that class. The benchmark uses between four and 516 bytes for constant data, with a mean of 124.70 and a standard deviation of 300.32. This high mean and standard deviation is due to the large spread of constant pool usage. The constant pool size passed is distributed quite evenly over the spectrum of context switches. Eight and 28 bytes are passed most frequently, making up just over 56% of all constant pool exchanges between partitions. The rest are evenly distributed between 4, 30, 264 and 516 bytes. Eighteen and 24 make up an inconsequential amount.

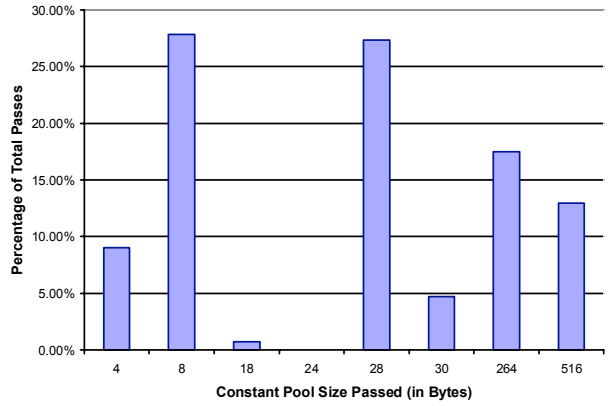


Figure 6: Constant Pool Size Passed Between Partitions

With the partitioning scheme considered here, the constant pool is not accessed by any of the instructions in the hardware partition. This indicates that the constant

pool may not need to be passed to hardware. This would increase performance of the partitioning strategy.

5.3 Stack

Figure 7 and Figure 8 outline the size of the stack passed between the two partitions. Unlike the constant data set size of local variables and constant pool data, the size of the stack changes during execution, thus the stack size passed to hardware and stack sized passed to software will be different and are considered separately.

The stack size being passed from hardware to software is outlined in Figure 7. The stack size being passed varies from zero to 39 bytes, with a mean of 9.59 bytes and a standard deviation of 6.26 bytes. Stack size of four, eight and twelve bytes dominate the rest of the stack sizes being passed, making up almost 55% of all stacks passed. The remaining 45% are spread evenly up to 22 bytes in length, the rest make up an inconsequential amount of stacks passed.

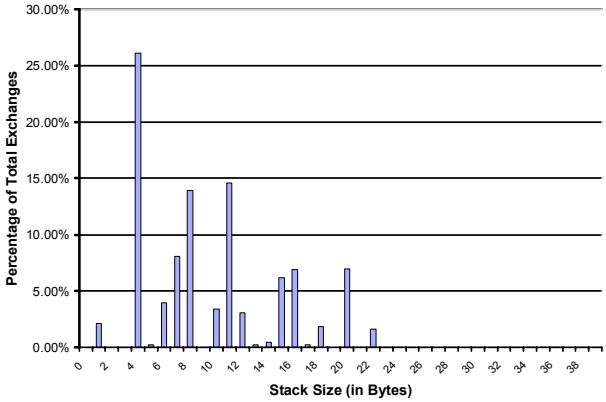


Figure 7: Stack Size Passed to the Software Partition

Since the stack is an integral part to execution it has to be passed to the software partition from the hardware partition. This is easily seen by the fact that all the hardware operations require the data on the stack in one fashion or another: to read from it; compare the values on the top; place values on it; or to remove values from it. Most of these operations change the state of the stack. However, since the stack size to be passed is relatively small, it will not be a major performance burden.

Figure 8 outlines the stacks being passed to the hardware partition from the software partition. The stacks vary in size from zero to 38 bytes in length, with a mean of 5.25 bytes and a standard deviation of 4.34 bytes. Five stack sizes dominate the sizes of the stacks passed to hardware: 1, 5, 8, 9, and 12 bytes. These stack sizes account for just over 92% of all stacks passed. The other 8% is spread quite evenly between the remaining stack sizes.

When comparing the stacks passed to hardware and the stacks passed to software it can be seen that the stacks passed to software are spread out over a larger range. The stacks being passed to software are also larger in general than those being passed to hardware; this is indicated by the differences in the means.

The hardware partition absolutely requires the stack to be passed to it, and the stack needs to be implemented in hardware. The hardware stack needs to support all sizes from 0 to 38 bytes. Thus the hardware partition requires room to allow the stack to grow during execution.

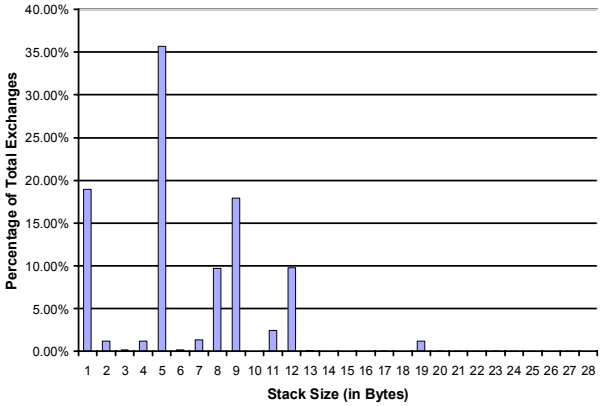


Figure 8: Stack Size Passed to the Hardware Partition

5.4 Method Size

The size of the methods that need to be passed from software to hardware are outlined in Figure 9. The method size passed to hardware is only considered because all methods exist in software to begin with and its state does not change during execution, thus it does not need to be passed to the software partition. The method size varies in length from one byte up to 419 bytes, with a mean of 192.85 bytes and a standard deviation of 186.88 bytes. Methods with a size of 419 bytes dominate the other sizes with just over 27% of all methods passed. The remaining method sizes are evenly spread in size between one and 170 bytes.

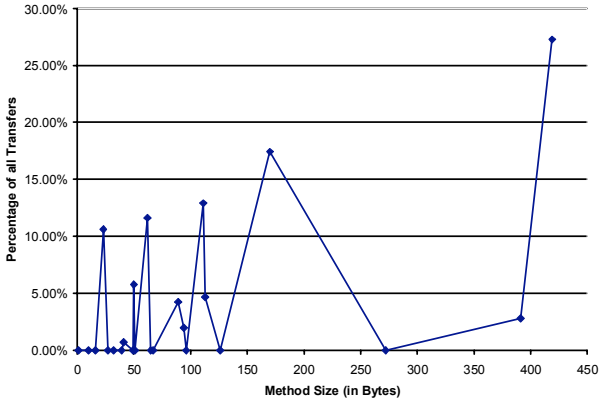


Figure 9: Method Sizes Passed to the Hardware Partition

Obviously all methods executed in hardware must have their opcodes passed to hardware. The large number of bytes required to pass the methods to hardware may present a performance bottleneck. One method to overcome this may be to load methods to a high speed ram that is local to the hardware partition. This would alleviate some of the communication issues for passing methods.

6 Processing vs. Communication Rates

Since communication between partitions is a major issue when considering a co-designed system, the amount of processing performed should be compared with the amount of communication required to pass the data required for execution. This will give an indication as to how well the partitioning scheme performs. Ideally a small amount of data will be passed to execute a large number of instructions.

Table 2 outlines the data passed from the hardware partition to the software partition vs. the amount of execution when in the software partition. The best, average and worst case scenarios are shown. In the best case only 8 bytes of data are passed for executing 6 instructions, making for 1.33 bytes communicated per instruction. The average case is 163 bytes passed for executing 2 instructions, averaging to 81.5 bytes passed per instruction executed. In the worst case 615 bytes are passed to execute only 1 instruction. The low number of instructions executed in each case can be attributed to the fact that only slightly over 13% of all instructions are executed in software.

	Data (in Bytes)	Instructions Executed	Bytes per Instruction
Best Case	8	6	1.33
Average Case	163	2	81.5
Worst Case	615	1	615

Table 2: Data Passed to Software Partition vs. Instructions Executed

Table 3 shows the data passed from the software partition to the hardware partition vs. the amount of execution when in the hardware partition. Again, the best, average, and worst case scenarios are shown. The best case has 9 bytes being communicated to the hardware partition in order to execute 813 instructions, making for just over $1/10^{th}$ of a byte passed per instruction executed. In the average case 346 bytes are communicated to execute 8 instructions, causing an average of 44 bytes per instruction. The worst case has only 1 instruction needing 1033 bytes of data for execution. The more efficient communication to the hardware partition can be attributed to the fact that just under 87% of all instructions are executed in hardware.

	Data (in Bytes)	Instructions Executed	Bytes per Instruction
Best Case	9	813	0.011
Average Case	352	8	44
Worst Case	1033	1	1033

Table 3: Data Passed to Hardware Partition vs. Instructions Executed

The data shows that in the worst cases the required amount of communication outweighs any performance gains presented by executing in hardware, thus context switches in the worst case, i.e. executing only one instruction, should not occur and execution should be left in software. The same cannot be considered for the worst case of execution in the software partition since software instructions cannot be implemented in hardware, while the opposite is not true. This data enforces the recommendation to implement a context switching prediction module to predict suitable context switches to hardware.

7 Conclusions

The previous sections propose a partitioning scheme for the Common Language Infrastructure instruction set and analyze its performance. The analysis performed in this paper shows that the preliminary partitioning proposed here performs well. The conclusions and observations regarding this partitioning are outlined in Table 4 and the recommendations for hardware implementation are outlined in Table 5. The major recommendation is to implement a context switching prediction module to predict suitable context switches to hardware. Since communication takes computing time, when a large amount of data needs to be passed to hardware for a small amount of execution, the payoff for switching to hardware is not great enough to justify a context switch.

Aspect	Sub-aspect	Observations
Performance of Partitions	Instructions Executed in the Software Partition	13.33% of all instructions executed are executed in software Average of 1.07 instructions executed consecutively.
	Instructions Executed in the Hardware Partition	86.67% of all instructions executed are executed in hardware. Average of 7.04 instructions executed consecutively
Context Switching	Instructions Generating Context Switches	63% of implemented hardware instructions cause context switches. 92% of implemented software instructions cause context switches.
Data Movement Between Partitions	Local Variables	Average of 5.61 local variables need to be passed to hardware. Up to 14 local variables need to be passed.
	Constant Pool	Average of 124.7 bytes need to be passed. Up to 516 bytes need to be passed.
	Stack	Average of 9.59 bytes passed to software. Up to 39 bytes passed to software. Average of 5.25 bytes passed to hardware. Up to 38 bytes passed to hardware.
	Method Size	Average size of 192.85 bytes. Up to 419 bytes need to be passed.
Processing vs. Communication Rates		Average of 81.5 bytes need to be communicated to the software partition per instruction executed. Up to 615 bytes need to be communicated to the software partition per instruction executed Average of 44 bytes need to be communicated to the hardware partition per instruction executed. Up to 1033 bytes need to be communicated to the hardware partition per instruction executed.

Table 4: Observations for Partitioning the CLR

Aspect	Sub-aspect	Recommendations
Performance of Partitions	Instructions Executed in the Hardware Partition	Use a context switch prediction module to predict suitable context switches to hardware.
Context Switching	Instructions Generating Context Switches	Pre-load variables, constants and arguments into hardware.
Data Movement Between Partitions	Local Variables	Does not account for a large amount of data, and thus is not of great concern.
	Constant Pool	Constant pool is not used by hardware instructions, and may be excluded from data passed to hardware.
	Stack	Stack must be communicated between partitions. Hardware must allow for stack to grow.
	Method Size	Load methods to high speed ram accessible by hardware.
Processing vs. Communication Rates		A context switch prediction module can alleviate some of the performance problems brought by switching to hardware in the worst cases.

Table 5: Recommendations for Implementation of HW/SW Co-designed CLR

References

[1] J. Miller. *Technical Overview of the Common Language Runtime*. Microsoft Corporation, 2000.

[2] K. B. Kent, H. Ma and M. Serra, *Rapid Prototyping of a Co-Designed Java Virtual Machine*, 15th IEEE International Workshop on Rapid Systems Prototyping, pp. 164-171, June 28-30, 2004.

[3] M. W. El-Kharashi, F. Elguibalay, and K. F. Li., *A Quantitative Study for Java Microprocessor Requirements. Part I: Instruction Set Design. Microprocessors and Microsystems*, vol. 24, no.5, pp. 225-236, September 2000.

[4] M. W. El-Kharashi, F. Elguibalay, and K. F. Li., *A Quantitative Study for Java Microprocessor Requirements. Part II: High-level Language Support. Microprocessors and Microsystems*, vol. 24, no.5, pp. 237-250, September 2000.

[5] Unknown. *Standard ECMA-355 Common Language Infrastructure (CLI)*. European Computer Manufacturers Association Standards, 2006.

[6] Rhys Weatherley. PNetMark FAQ. Southern Storm Software, http://www.southern-storm.com.au/doc/pm_faq.html, [Accessed 2002-12-11].

[7] David Barker-Plummer. Turing Machines. *Stanford Encyclopedia of Philosophy*, http://plato.stanford.edu/entries/turing-machine/ [Accessed 2005-4-7].

Dimensionality Reduction for the Control of Powered Upper Limb Prostheses

Klaus Buchenrieder
Fakultät für Informatik
Universität der Bundeswehr München
D-85577 Neubiberg, Germany
E-mail: Klaus.Buchenrieder@unibw.de

Abstract

Myoelectric prosthesis control is difficult to achieve, because the optimal selection of features from a set, extracted from a myoelectric signal, is known as being a very challenging problem in prosthetic control. Dimensionality reduction means to retain MES information, that is important for class discrimination and to discard irrelevant data. Dimensionality reduction strategies can be categorized into feature selection and feature projection methods according to their objective functions.

This work brings forward a statistical cluster analysis technique, which we call the "Guilin Hills Selection Method". The goal of our efforts is to control electrically powered upper limb prostheses with a minimum number of sensors and a low-power processor. The paper introduces the method and brings forward exemplar results for the selection of two time-domain features. With the proposed method, we can clearly distinguish four hand-positions from myoelectric signals of two sensors. The method also suggests feature combinations, that lead to less robust classifications but require fewer sensors.

1. Introduction

Today, many myoelectric systems for the control of a prosthetic limb, such as a hand, an elbow or a wrist are commercially available. These systems extract control information, such as amplitude, rate of change, etc., from myoelectric signals (MES). Research in MES processing dates back to the early 1960s [1] and many of the techniques developed in the early years are still used in medical products today. A conclusive summery of the advances in upper limb prosthesis and microprocessor control was brought forward by Lake and Miguelez [2]. Muzumdar provides in his book an extensive survey of control methods, prosthetic implementations and clinical applications [3]. Most control systems are based on Hudgins' work and use a simple multilayer perceptron (MLP) as classifier for a set of time-domain features extracted from the MES [4,5,6].

Based on these fundamental ideas, a prosthesis control system was developed at the *Universität der Bundeswehr München* [7]. The UniBw-Hand controller is different from the original approaches with respect to feature classification, training method and comfort for the user. For the operation of the system, we distinguish a *training*- and an *operative- mode.*

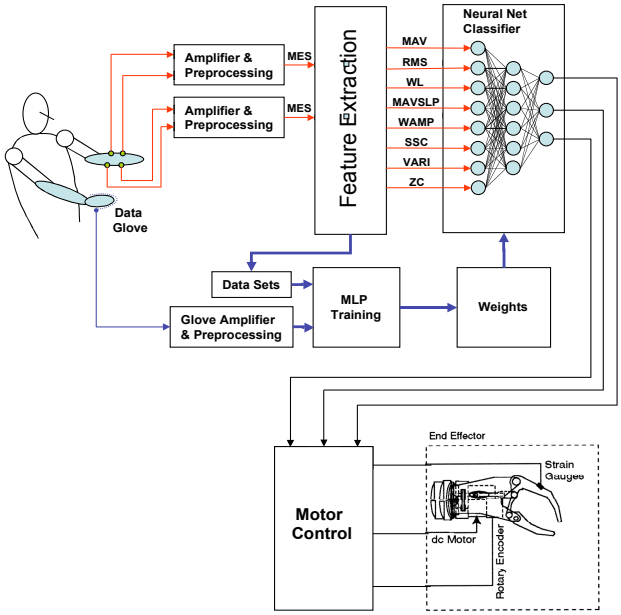


Figure 1: The UniBw-Hand control system for powered upper limb prostheses.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

During the *training-mode*, users exercise predefined hand and arm motions while MES data are measured at the remaining predominant muscles, e.g. on the patients forearm. The data sets acquired are used to train the multilayer perceptron, which interprets the MES in the operative mode. In contrast to traditional training methods, where users are asked to solely flex muscles, we also measure precise motion of the opposite hand with a data-glove. The use of the glove allows to exactly relating true hand gesture and movement to the MES of the impaired side. A set of time-domain features is extracted from a transient burst of the MES. Since the signals are nonlinear and stochastic, it is difficult to decide which signals to chose. Normally, the selection is based on experimentation, prior experience, or on published combinations known to provide good results for certain combinations of hand-positions with a muscle. Selected features are then extracted and presented to, in our implementation, a neural net classifier. The correlation of movement and MES ensures precise training data and accurate weights for the operative mode. A block diagram of the control scheme is shown in Figure 1.

In the *operative-mode*, the MES is measured and preprocessed exactly as in training mode. With the weights obtained during the training phase, controls for the motor are generated. Amplifiers of the motor control provide steering signals for the prosthetic hand device. A current limiter for the motor and sensors provide sufficient feedback to the motor control and ensure a force regulated and slippage reduced grip.

Feature selection is known to be one of the most difficult problems in myoelectric signal processing. It is of fundamental importance, because a proper combination of features is instrumental to achieve a good classification and optimal results. The choice is especially crucial, when several hand-positions and transitions between positions must be distinguished from MES based on a small set of functional muscles.

2. Dimensionality Reduction

In myoelectic signal processing, dimensionality reduction is the process of feature selection and feature reduction, so that important information for best class discrimination is retained and superfluous information discarded. Fodor provides a survey of dimensionality reduction techniques [8].

Dimensionality reduction methods are distinguished into *feature selection* and *feature projection* strategies.

Feature selection methods try to find the best subset of an original, initial set of features. In the context of myoelectric signal processing, feature selection is based on Euclidian distance calculations and a class separation criterion. Supervised methods rank the features according to available class membership information. Generally, the features are chosen, so that the distance of the "centers of mass" for the sample-clouds of hand-positions, obtained during a training phase, are maximized.

In contrast, feature projection methods attempt to determine the best combination of original features through a mapping of the original multidimensional feature space into a space with fewer dimensions. Typically, an original feature space is transformed with a linear transformation, e.g., with principal component analysis (PCA), that provides a means of unsupervised dimensionality reduction, as no class membership qualifies the data when specifying the eigenvectors of maximum variance [6].

3. The Euclidian Distance Calculation Method

In the initial development of the controller for the UniBw-Hand, we employed the state-of-the-art Euclidian distance calculation method to select the features. Hereby, we followed the standard approach and assumed a uniform weight for the data samples. As an example consider Figure 2. It shows a three-dimensional feature map indicating clouds of samples for wrist-flexion with green-, wrist-extension with blue-, and fist with red-data-points. For a sensor placed at the flexor carpi radialis muscle, we chose the features: RMS Quotient (RMS), Zero Crossings (ZC) and Waveform Length (WFL), to achieve a maximal distance for the three "centers of gravity".

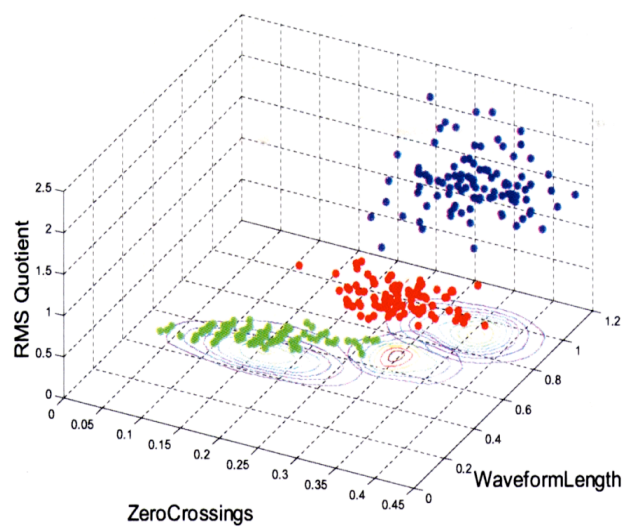


Figure 2: 3-D Feature map for a sensor placed at the carpi radialis resulting in a proper separation of the three hand-positions.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

The footprint (down-projection) on the ZC-WL plane indicates, that undesired overlaps in the features space exist, when only the dimensions ZC and WL are considered. Therefore, an RMS feature is added for an optimal separation. With this method we achieved very good classification results.

4. The Guilin Hills Selection Method

The Euclidian distance feature selection method becomes unsuitable when extensive intersections of sample-clouds exist. Since individual muscles contribute simultaneously to several hand-motions or -positions, a sample point, as found in the strongly clustered center of a cloud, classifies a hand-position better than a more distant point. As an example regard the flexor carpi radialis. It is flexed when adopting a fist, a radial flexion and some other positions. When all points contribute to the center calculation with identical weights, outlier points shift the center of gravity away from the region for an optimal classify-cation. Therefore, features must be chosen, so that the centers of probability, the hills, are maximally distant.

The Guilin Hills selection method was inspired by the beautiful landscape of the hills and the Li river close to Guilin in southern China, and the observation, that sets of data points, for a (time-domain) feature are normal distributed. This has been confirmed by testing more than one hundred sets of training data, which were all found to be mesokurtic, with $\Upsilon_N \approx 0$.

Based on this, we can represent a training data set for a feature of a specific muscle and a certain hand-position as a tuple of mean (a_n) and standard deviation (d_n). The bell-shaped curves in Figure 3, represent the standard distributions of WFL features, derived for the flexor carpi radialis from four sets of 100 training samples, each for the hand-positions (left to right): radial flexion, wrist-extension, fist, and wrist-flexion.

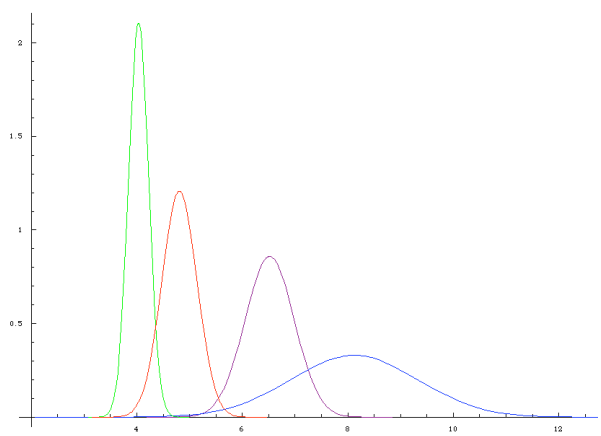


Figure 3: WFL distributions for four hand-positions calculated for the flexor carpi radialis.

During the operative phase of prostheses, features, called measurements, are continuously derived from the MES and classified based on the training data. An ideal association with a position is possible only when the probability density functions are well separated. Hence, features must be chosen, so that either: the integral under the combined curve is minimal within, i.e. three, standard deviations of the respective means, Figure 4a; or: the combined density function at the intersection is minimal, Figure 4b.

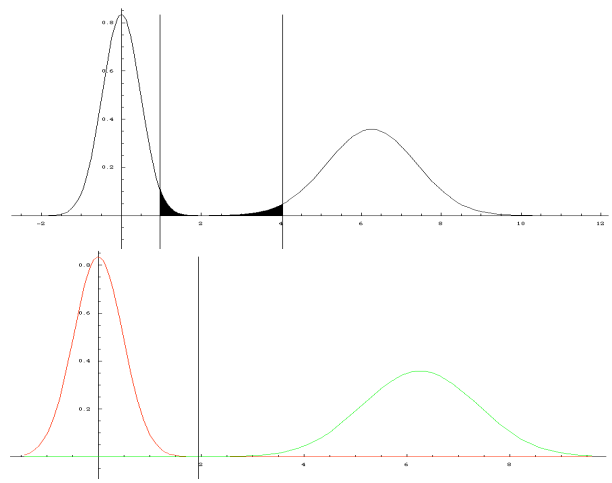


Figure 4: (a) The upper graph shows the combined 3σ areas, (b) the grey vertical marks the valid intersection point x_{int}.

Here, we calculate the combined density function:

$$CombinedNvd\left(x_{\text{int}+}, x_{\text{int}-}, a_1, d_1, a_2, d_2\right) =$$

$$\max\left(\begin{array}{l} \dfrac{1}{\left(\sqrt{2\pi}d_1\right)e^{\frac{\left(x_{\text{int}+}-a_1\right)^2}{2d_1^2}}} + \dfrac{1}{\left(\sqrt{2\pi}d_2\right)e^{\frac{\left(x_{\text{int}+}-a_2\right)^2}{2d_2^2}}}, \\ \dfrac{1}{\left(\sqrt{2\pi}d_1\right)e^{\frac{\left(x_{\text{int}-}-a_1\right)^2}{2d_1^2}}} + \dfrac{1}{\left(\sqrt{2\pi}d_2\right)e^{\frac{\left(x_{\text{int}-}-a_2\right)^2}{2d_2^2}}} \end{array}\right);$$

at the intersection point x_{int}, for two feature tuples (a_1,d_1), (a_2,d_2) as:

$$Intersection\left(a_1, d_1, a_2, d_2\right) = x_{\text{int}+}, x_{\text{int}-} =$$

$$\frac{1}{d_1^2 - d_2^2}\left(\begin{array}{l} a_2 d_1^2 - a_1 d_2^2 \pm \\ \sqrt{\left(\begin{array}{l} a_1^2 d_1^2 d_2^2 - 2a_1 a_2 d_1^2 d_2^2 + a_2^2 d_1^2 d_2^2 - \\ 2d_1^4 d_2^2 \log\left(\dfrac{d_2}{d_1}\right) + 2d_1^2 d_2^4 \log\left(\dfrac{d_2}{d_1}\right) \end{array}\right)} \end{array}\right);$$

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Note, that only one of the two resulting intersection points is valid, because it lies between the "hills" and can be used to select the feature. In the following we refer to the combined density at the intersection point simply as *crossover*. As Figure 7 shows, the zero crossing feature (ZC) for the flexor carpi radialis provides the best differentiation, however, overlap still exists. The result can be improved only by adding a carefully selected, additional feature as second dimension.

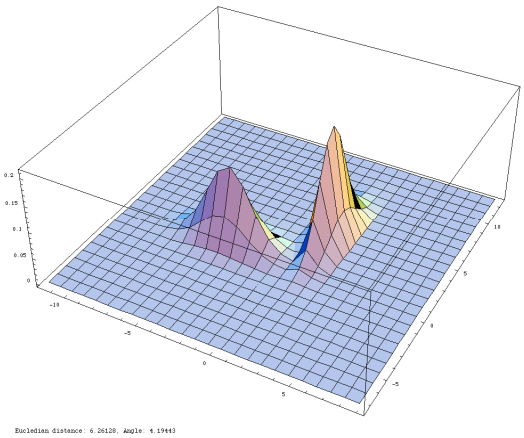


Figure 5: Combined probability function for two positions and two features.

The surface plot in Figure 5 shows the combined probability function for two hand-positions and two features. The contour plot 6a, reveals a clear separation of the elliptic footprints, which were constructed from the normalized standard deviations. A good approximation of the cross-sectional cut, required to determine the crossover, is shown in Figure 6b. The new standard deviations (d_{cs1},d_{cs2}) are derived from the angle φ between the centers of the ellipses with the parametric equation:

$$x = d_1 \cos\varphi; \quad y = d_2 \sin\varphi; \Rightarrow d_{cs1} = \sqrt{\frac{d_1^2 d_2^2 \sec\varphi^2}{d_1^2 + d_2^2 \tan\varphi^2}};$$

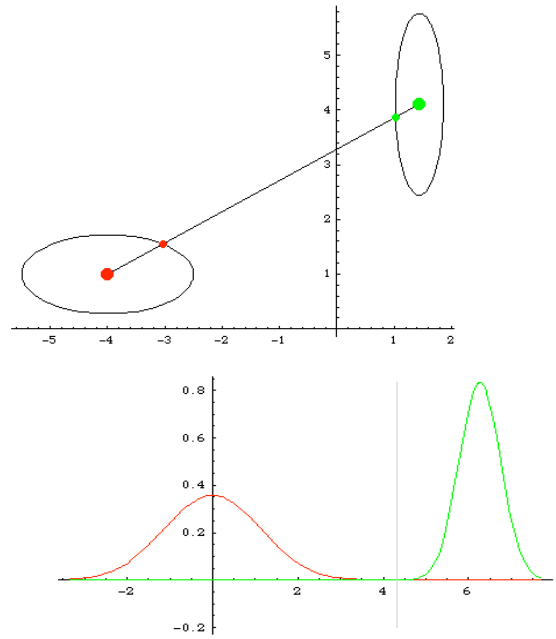


Figure 6: (a) cross sectional contour plot, constructed as baseline for the combined density function in (b).

The Euclidian distance of the points, where the semi minor and semi major axes of the ellipsis intersect, serve as new means (acs1, acs2).

When more than two positions are to be distinguished, the mutual distance between all the hills should be as large as possible. To determine the optimal combination of features, one must calculate the largest, so called worst-case, crossover for all combinations of features with sensor pairings. The desired combination obviously is the pairing with the lowest value for the worst-case crossover.

At the first glance, this method seems computationally expensive since all combinations must be explored. In fact, only a fraction of the combinations must be considered, since pruning and backtracking reduces the solution space exceptionally well.

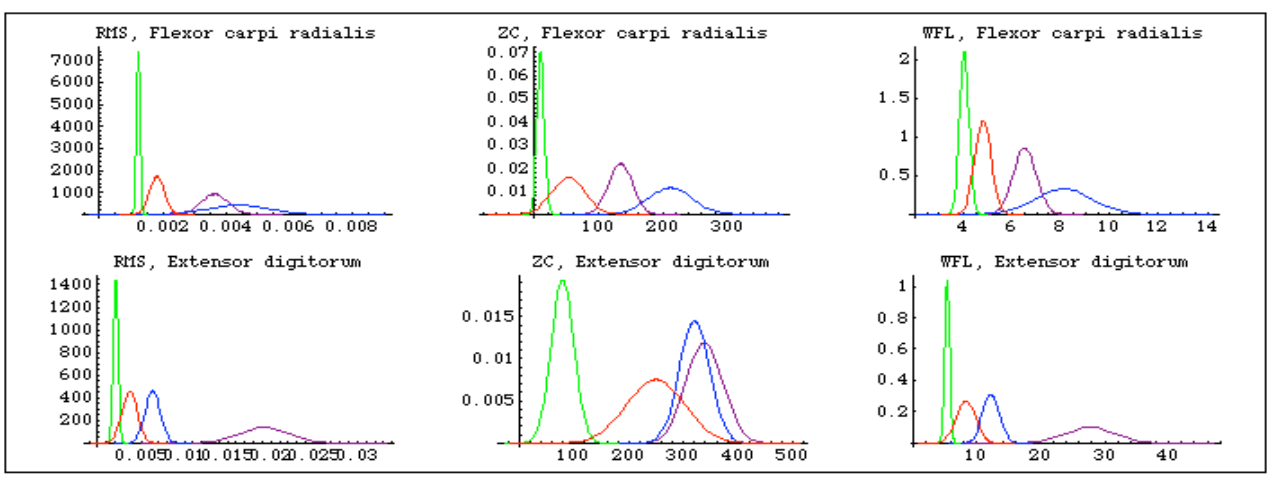


Figure 7: Display of four hand-positions over three features and two muscles.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

5. Experiments and Results

The Guilin Hills feature selection method was tested with numerous data sets. For data recording, we used four active bipolar skin-surface electrodes and a separate common (ground) electrode, as shown in Figure 8.

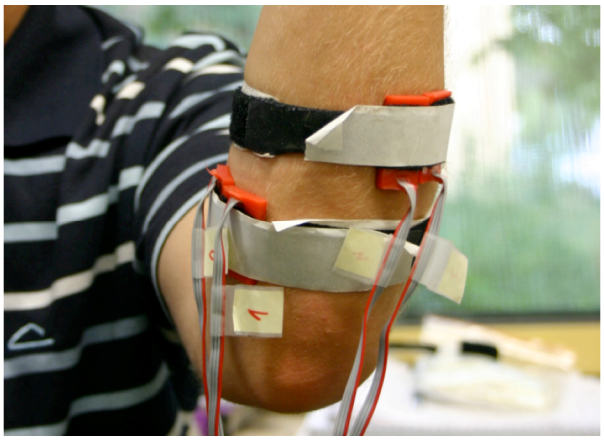


Figure 8: Application of four active bipolar electrodes for data recording and prostheses control.

Each bipolar electrode consists of two stages, an active pickup, with a gain of G_1=10, and an amplifier with a gain of G_2 = 20 and an active filter with a pass-band window from 70 Hz to 400 Hz. Following the preprocessing stage, the four channels were sampled with a PCIe-6251 Express M Series multifunction dedicated data acquisition board (DDA) from National Instruments. We chose an accuracy of 16 Bit at a sampling rate of 1024 Hz and found this being an ideal setting. The raw data was extracted from a sampling interval of three seconds. In the feature extraction stage, implemented with the Labview Software environment Version 8, also from National Instruments, we extracted the standard time-domain elements: mean absolute value (MAV), root mean square (RMS), waveform length (WL), mean absolute value slope (MAVSLP), Willison amplitude (WAMP), slope-sign changes (SSC), integrated absolute value (IAV), variance of EMG (VARI) and zero crossings (ZC). The extracted elements were then stored as feature files for subsequent processing with the Mathematica 5.2 environment.

For this contribution, we present exemplar results for the selection of two features out of three time-domain elements for two muscles to distinguish four hand-positions. The selected features are suitable for our experimental UniBw-Hand controller, that employs a three-layer neural-net classifier, implemented in Labview [9,10]. The elements chosen for the demonstration are: RMS, WFL and ZC for the *flexor carpi radialis* and the *extensor digitorum*.

The root mean square, also known as the quadratic mean, is a statistical measure of the magnitude or strength of the muscle contraction. The RMS for N samples $\{x_1,\ldots,x_N\}$ is:

$$RMS = \sqrt{\frac{1}{N}\sum_{i=1}^{N} x_i^2}\,.$$

The waveform length provides information about the frequency range and the amplitude of the signals [5]:

$$WFL = \sum_{i=1}^{N}\left|x_i - x_{(i-1)}\right|.$$

Zero-crossings denote the number of sign-changes along a signal within a predefined time window. The threshold S is introduced to reduce noise effects. ZC provides information about some frequency related properties [11]. For two consecutive samples x_k and x_{k+1}, over the whole time frame holds:

$$ZC = \sum_{k=1}^{N} g_{zc}(x_k) \quad with$$

$$g_{zc}(x) = \begin{cases} 1 \; if \;\; \mathrm{sgn}(-x_k \times x_{k+1}) \wedge \left(\left|x_k - x_{k+1}\right|\right) \geq S, \\ 0 \; otherwise \end{cases}$$

whereby

$$\mathrm{sgn}(x) = \begin{cases} true \; if \;\; x \geq 0, \\ false \; otherwise \end{cases} .$$

As positions, we have chosen radial flexion, wrist-extension, wrist-flexion, and fist for five reasons: first, all patients can easily perform these motions regardless of their limitation; second, the MES of the three positions and transition motions can be distinguished with only two differential myoelectric sensors and standard classification methods; three, the muscle action relating to a wrist-flexion, a fist and a wrist-extension can be unambiguously related to an end-effector control of close, gripper-force increase and open action; four, the detection of the co-contraction of the two muscles can be employed to, e.g., switch the mode of the end-effector from the grip to a wrist-rotation mode; and last, we have implemented controllers for these positions and can compare the results of the feature set obtained with the Guilin Hills method with previous findings.

After recording of the 24 raw data sets, the features were calculated for each hand-position and the two muscles. Illustration 7 shows, the one-dimensional result graphs for each muscle and feature. As already explained, one feature-dimension is not sufficient because the probability functions overlap. In the following, combinations of features are explored and

distance matrixes calculated. As one such combination consider Figure 9. It shows the normalized surface plot for the combination RMS and WFL with the flexor carpis radialis, labeled as muscle 1. The two hills intersecting in the center of the plot correspond to the positions of wrist-flexion and fist.

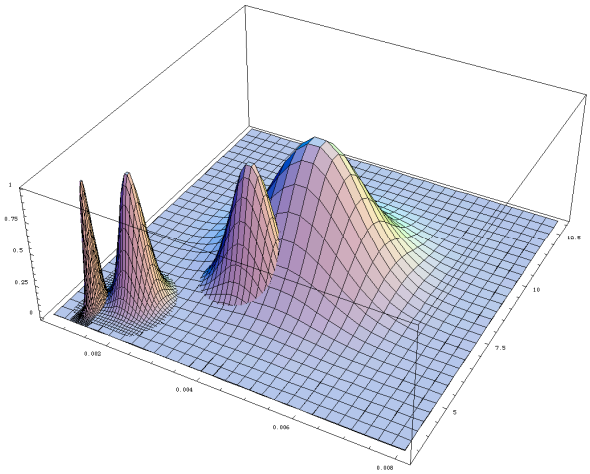


Figure 9: Normalized surface plot for the combination RMS1-WFL1.

The large, central hill represents the wrist-flexion rms31-wfl31 and has a large standard deviation. Since the plot is normalized, it appears as a high peak but in reality it is a rather unimpressive mound. To provide a visual clue about the relative height and the standard deviation, the same grid resolution is chosen for all individual mountains. For the rms31-wfl31 the grid is comparatively coarse.

Table 1: RMS1-WFL1 Crossover Matrix

Wrist-Flexion rms31-wfl31	Wrist-Extension rms41-wfl41	Radial-Flexion rms61-wfl61	RMS1-WFL1
0.25855	$4.6172 \cdot 10^{-10}$	.00336685	**Fist** rms21-wfl21
	.000029509	.00535947	rms31-wfl31
		.0671984	rms41-wfl41

Table 2: Feature Selection Matrix: RMS-WFL-ZC

RMS2	WFL1	WFL2	ZC1	ZC2	
1.3952	.258550	.0201346	.00440842	.0210817	RMS1
	.208677	.0501229	.00391864	.00391293	RMS2
		.0405795	.00352604	.02322640	WFL1
			.00335150	.00446167	WFL2
				.00453696	ZC1

The RMS1-WFL1 crossover matrix, in Table 1, shows the wide reaching influence of the wrist-flexion mound. For the RMS1-WFL1 combination, the central situation is clearly reflected by the worst-case crossover of 0.25855. After all crossover matrixes are calculated the worst-case results are combined in the feature selection matrix: RMS-WFL-ZC-42, Table 2.

The minimal crossover of 0.00335150 corresponds to the feature combination ZC1-WFL2. This selection provides the best separation for the four hand-positions when using two sensors. The second best solution, with a worst-case crossover of 0.00352604, corresponds to ZC1-WFL1. This combination requires only a single sensor. The Guilin Hills surface plot for both solutions, ZC1-WFL2 and ZC1-WFL1, is depicted in Figure 10a and 10b.

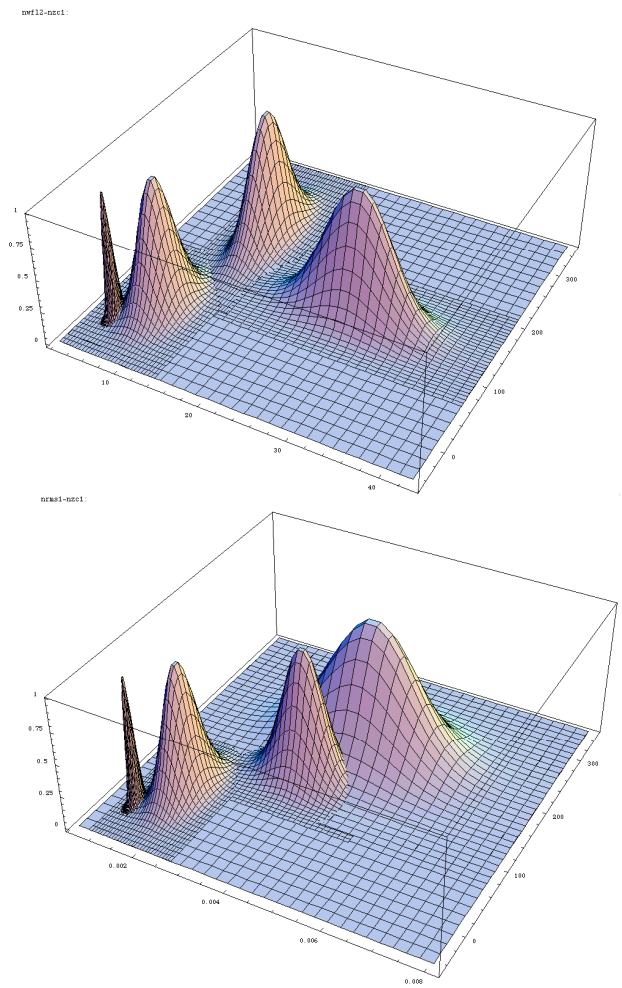


Figure 10: Normalized results for the selected feature combinations: (a) ZC1-WFL2 and (b) ZC1-WFL1.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

6. Conclusions

As already stated in our previous work [7], the main purpose for the development of the UniBw-Hand is to educe new methods and to build a demonstrator that helps patients to be more independent. Since we develop baseline technology, it is left to our industrial partners to evolve medical products with our findings and to commercialize the results. The long-term goal of the UniBw-Hand project is to devise a safe and reliable control system for low-cost hand prostheses. The envisioned system will only require a minimum of MES sensors for grasp, pick and place actions, while allowing for simultaneous rotation of the artificial hand.

Even though many improvements have been suggested for the advancement of artificial hand control, the research in the field remains challenging, in that only few problems have been satisfactorily solved or solutions have been found resulting in affordable solutions for prosthetic products.

In this paper, we present the Guilin Hills feature selection method, which significantly improves the classification of MES signals and can, in some cases, be used to reduce the number of required sensors. Whether a prosthesis control with a minimum number of sensors is practicable, reliable and safe requires extensive tests with a commercial prosthetic device such as a MYOBOCK™ System Electrohand [12] from Otto Bock Healthcare GmbH. These tests are part of the ongoing research.

When working with the presented method, we gave up on solely relying on traditional combinations of features or muscles for the classification of sets of hand-positions. Current results show the potential to use features from fewer, still intact muscles to discern related positions. At this state of the research, it is however unclear how muscle fatigue changes the positions and standard deviations of the probability functions involved.

The preprocessing of the MES and the extraction of features manifested the need for a more advanced technique to remove noise and artifacts from the raw MES. For this reason, future work will include the development of autocorrelation filters embedded in the preprocessing and amplification stage as well as multi-electrode-arrays. Major improvements in the MES classification have also been achieved with time-frequency domain features deduced with wavelet-transformation techni-ques. Efforts will be made to apply the Guilin Hills selection method to features out of the time-frequency domain and in combination with fuzzy-classifiers.

Acknowledgements

The author would like to thank Mr. Dimtry Filippov for recording and preprocessing hundreds of data sets of static hand-positions for testing the Guilin Hills feature selection method. The author also expresses his gratitude to Mr. Stefan Merz for the sampling of data-sets capturing transitions between hand-positions and the refinement of feature extraction procedures in Labview.

7. References

[1] Horn, G. "Electromyographic signal produced by muscle movement, controls grasp of prosthetic fingers". *In Electronics*, October, 1963, pp. 34-36.

[2] Lake, C. and J. Miguelez. "Comparative Analysis of Microprocessors in Upper Limb Prosthetics". In *American Academy of Orthotists and Prosthetics Journal of Prosthetics & Orthotics* 15, No. 2, 2003, pp. 48-65.

[3] Muzumdar A., *Powered Upper Limb Prostheses*, Springer-Verlag, Berlin and Heidelberg Germany, 2004.

[4] Hudgins, B.S., "A novel approach to multifunctional myoelectric control of prostheses", *Ph.D. Thesis*, Department of Electrical Engineering, University of New Brunswick, Frederiction, New Brunswick, Canada, 1991.

[5] Hudgins, B., P. A. Parker, and R.N. Scott, "A New Strategy for Multifunction Myoelectric Control", In: *IEEE Transactions on Biomedical Engineering*, 40, 1993, January, pp. 82-94.

[6] Englehart, K., B. Hudgins, P.A. Parker, and M. Stevenson, "Classification of the Myoelectric Signal using Time-Frequency Based Representations", Special Issue of *Medical Engineering and Physics on Intelligent Data Analysis in Electromyography and Electroneurography*, Summer, 1999.

[7] Buchenrieder, K.J., "Processing of Myoelectric Signals for Control Assistance of Multifunctional Hand Prosthesis", In *Proceedings of the International Mediterranean Modelling Multiconference*, Piera, LogiSim, Barcelona, Spain, September, 2006, pp.653-657.

[8] Fodor, I.K., "A Survey of Dimension Reduction Techniques", *Report of the U.S. Department of Energy*, Report Nr. UCRL-ID-148494, Lawrence Livermore National Laboratory, May, 2002.

[9] Sippel, M. 2005. "Verarbeitung myoelektrischer Signale zur Erkennung von Bewegungen und Stellungen einer Hand", *Thesis*, UniBwM-ID 40/2005, Neubiberg Germany, 2005.

[10] Trautsch, R.. "Biosignalverarbeitung zur Steuerung einer Handprothese", *Thesis*, UniBwM-ID 39/2005, Neubiberg Germany, 2005.

[11] Zecca, M., S. Miecera, M.C. Carrozza, and P. Dario, "Control of Multifunctional Prosthetic Hands Processing the Electromyographical Signal", In *Critical Reviews in Biomedical Engineering,* 30, 2002, pp. 302-311.

[12] MyoBock, 2003. Otto Bock Healthcare [online]. Available: http://www.ottobock.com/ seen 8/5/2006.

Computer Based Psychotherapy for Treatment of Depression and Anxiety

Byron Purves and David Purves
Self Help Solutions, Inc.
byron.purves@bluesbegone.com

Abstract

This paper describes a new computer based delivery method of Cognitive Behavioral Therapy for relief of depression or anxiety. Blues Begone combines established psychological principles with a novel implementation that makes depression recovery available to many more people than face-to-face therapy, and without the side effects of drugs. A flexible architecture makes possible significant variation in content without software change.

1. Introduction

Automation has replaced many tasks that have traditionally been performed by humans. It is now accepted in many disciplines to have some of the mental labor done by computers.

One area that so far has been scarcely impacted by growth in computing has been psychological treatment. No doubt there have been good reasons for this. Perhaps the human mind was considered too complex and delicate for therapeutic contact with software. Some twenty-five years ago a toy therapist program was given a great deal of publicity, and some observers ascribed a great deal more intelligence to *Eliza* than it (she) actually deserved. [1] Another obstacle may have been the popular picture of a psychotherapist analyzing a patient for weekly sessions over a period of many years [2]. Perhaps psychologists were not able to formally describe their methods to a degree that would allow software engineers to imitate them. And then there is the problem of patient acceptance. Special significance has always been attached to the interaction of a patient with another human being who cares [3].

Some useful advances have been made quite recently that have begun to change the landscape for computer based psychotherapy. Contrary to the expectation that human interaction was necessary for psychological progress, studies on reading self-help books have shown that depression and anxiety can be alleviated by reading certain kinds of books [4]. Another important advance in psychotherapy itself was the introduction of Cognitive Behavioral Therapy (CBT) [5]. CBT did not require many months of patient "analysis." On the contrary, many people could be helped significantly in a dozen or fewer sessions [6]. CBT does not require the detailed review of a patient's history, but can work effectively by means of a scheme of information and exercises. In fact, CBT practitioners often describe their approach by representing the human brain as a computer.

2. Cognitive Behavioral Therapy

The core concept of CBT is that, for example, unpleasant feelings of depression are preceded by thoughts that trigger the unpleasant feelings, and that the thoughts themselves are triggered by a situation or event. CBT recognizes that the initial thought may not be completely valid, and thus the consequent feeling is inappropriate to the situation or event. The unpleasant feeling associated with, for example, depression, can be relieved by recognizing the distortion in the triggering thought and correcting it. Skill in recognizing and correcting damaged thoughts can be gained by learning about the process and by practicing its application. When we describe CBT therapeutic approach in these terms, it becomes clear that a computer can be used to facilitate the relevant education and practice.

A few early products have now appeared that support this automation. The first major application was *Beating the Blues* from Ultrasis [7]. This is a multimedia application. So far, it has been applied in British National Health Service clinical settings. Patients visit a central location once a week where they work on the computer for about an hour. A human assistant is available to help with matters like logging on and using the package. Positive outcome data has been reported, and as a consequence, the British National Institute for Clinical Excellence (NICE) has recommended computerized CBT [8].

Some other products applying CBT have become available, including for example, *Cope [9]* and *Overcoming Depression [10]*.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

3. The problem of depression

Depression is a major health problem. The World Health Organization (WHO) estimates that depression is the largest cause of disability in the world, and the fourth largest burden of disease [11]. In the United States it is estimated that around 10 percent of the population suffer from depression in any year [12]. Most depression remains untreated, resulting in poor quality of life. A majority of people do not in fact seek treatment because of unavailability of treatment resources, social stigma associated with mental health problems, failure to recognize that depression can be treated, or the high cost of psychological treatment.

We developed Blues Begone as a means of alleviating some of these deficiencies.

4. Architectural Approach

From the problem statement above we can deduce a set of high-level design requirements.

- Accessibility for wide range of people (including people not especially computer-savvy)
- Use of established CBT principles
- Confidentiality/privacy
- Low cost
- Measure degree of depression, and track progress
- Organized educational materials
- Practice at recognizing and correcting faulty thinking

The kind of attention to detail and rigor required in software development are in tension with the emotional focus and concern with human problems needed by a psychotherapist. We should anticipate a major challenge in operationalizing the human to human interface, eliciting requirements and validating a design. In our case, Blues Begone was developed by a pair of brothers: one a practicing psychologist, the other a system and software engineer. This teaming arrangement made feasible what otherwise might well have been simply too difficult.

Our first two prototypes were rejected. The first prototype tried to make the computer look like a therapist, superficially based on the Eliza model. While this attempt was helpful in discovering some of the data structures needed to capture relevant user information and relate it to necessary questions, the complexity and required vocabulary simply became too heavy. Nor could we be even sure that it would ever work.

The second prototype did not have a satisfactory user interface and was not sufficiently flexible.

We accepted the basic approach of the third prototype, although we made major changes and additions during development. For example, we did not initially plan to use synthetic voices, but rather to use video clips with actors. The decision to change was driven by cost, lead-time and inflexibility. In order to make a content change, it was necessary to bring together the same actors and re-film. It was much more cost-effective to change the content in a text file. Although speech recognition is included in the design, we found that user calibration was too difficult for many users in our population.

5.Architectural Issues

Conceptually, the therapeutic process is a feedback control loop, as shown in Figure 1. The therapist evaluates the client, decides on a course of treatment and begins to apply it. As the therapy proceeds, new information may result in a modification of the treatment regimen. This appears to be a simple feedback control loop. In practice there is considerable noise in measurement of the control error.

In Blues Begone, the primary control error is the score obtained using a depression measurement instrument. The instrument we use is called the Purves Depression Questionnaire (PDQ).

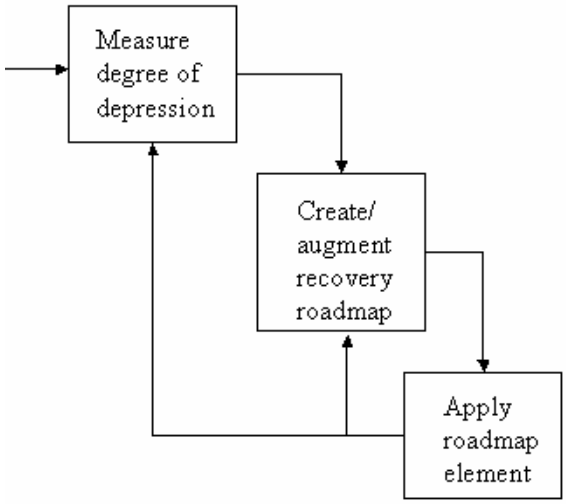


Figure 1 Functional Architecture

In order to make this measurement as part of the feedback control loop, we needed to create a new tool. This tool although new had to measure degree of depression in a way that correlates with prior methods. The practical definition of depression is defined by the American Psychiatric Association (APA). This is not a

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

quantitative definition [13], but any measurement must be compatible with it. Prior art used paper-based instruments including, for example, the Beck Depression Inventory (BDI-II) [14] and the CES-D [15]. In an early development phase we ran a group of 39 graduate students through the PDQ, BDI-II and CES-D, obtaining correlations with those instruments of 0.81 and 0.78 respectively. The correlation between BDI-II and CES-D was 0.70. Since measurement of depression is a hypothetical construct, we can conclude from these correlations and the relationship of its questions to the APA definition that the PDQ is measuring the right kinds of factors.

In addition to measuring degree of depression, the initial use of the PDQ includes some questions that deal with particular ancillary factors like alcohol or drug use, bereavement, self-esteem, etc. The combination of personal demographic data, PDQ score and additional answers give Blues Begone enough data to prepare an initial Roadmap to Recovery.

At base, Blues Begone includes a collection of 30 episodes. Each episode is not rigidly specified but rather is defined in terms of a scripting language. The scripts accommodate contingencies to respond to PDQ answers, and permit tailoring of the program as it appears to a user. After completion of the first PDQ, the scripts for each episode are executed, and the corresponding initial Roadmap to Recovery is computed. An illustrative fragment of a recovery roadmap is shown in Figure 2.

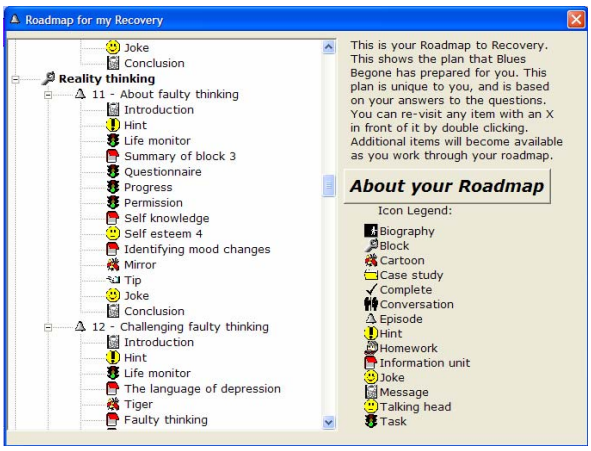


Figure 2 Recovery Roadmap Fragment

As the user proceeds through the roadmap of thirty episodes, the content is likely to be further modified to reflect observed need. User data is maintained in an encrypted database.

6. Building Blocks

Each object in the roadmap is one of the following kinds:

6.1 Information Unit

This is actually an HTML document, however it is retrieved strictly from the operations disk using a dedicated browser that appears on the Blues Begone desktop without recourse to the internet. We do it this way so that users do not need to be concerned about the possibility that data they enter may somehow be visible to others. Blues Begone includes about 90 of these Information Units. Usually they include a variety of illustrative cartoons. So far they do not include active content to avoid collisions with the user security apparatus.

6.2 Task

Tasks are very specific exercises aimed at giving the user practice in detecting or correcting the kinds of errors in thinking that lead to depression or anxiety.

Each task is unique and individually coded. Blues Begone includes about 30 of these.

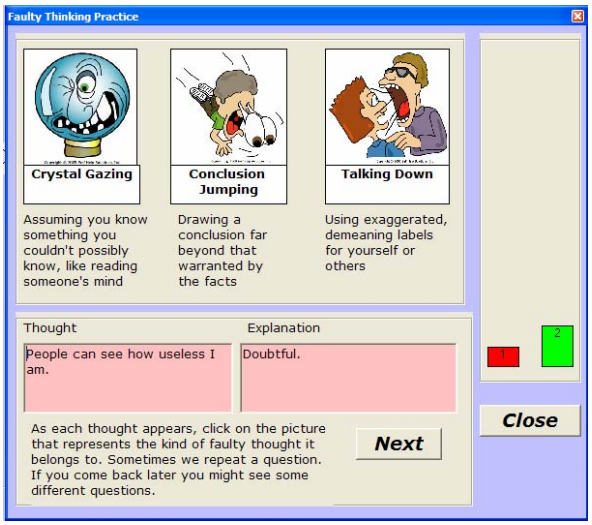


Figure 3 Sample Task – Faulty Thinking

6.3 Talking Head

A Talking Head consists of a cartoon head with multiple mouth and eye arrangements synchronized with a speech synthesizer. We use synthetic voices compatible with the SAPI 5 [16] standard. SAPI 5 allows the speech engine to generate events on word, viseme and utterance boundaries, so that the head can,

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

for example, match mouth shape to the sound currently being made. The text to be spoken is in the form of an XML document that passes through a pre-processor before being sent to the speech engine. The pre-processor replaces tags with things like the user's name, time-of-day-aware greetings, PDQ score, etc. Talking Heads can bring up other tasks, Information Units or Cartoons. One application uses a Talking Head to narrate a set of charts as they are cycled under script control. This flexibility means that the Talking Head can be quite personal and context aware in talking to the user. The speech script allows one particular talking head to be replaced by another during the speech. Blues Begone has a repertoire of three cartoon male heads (two of which look vaguely like the authors) and two female ones. While the Talking Head talks, the text being spoken is shown below the head cartoon. Blues Begone includes about 60 Talking Heads.

6.4 Conversation

A conversation is like two Talking Heads side by side carrying on a conversation with one another. Usually this is a conversation between a (fictional) therapist and a (fictional) client. Again, the underlying definition is an XML document with run time resolution of tags. Figure 4 shows a view of conversation heads (they look a little bit like the authors).

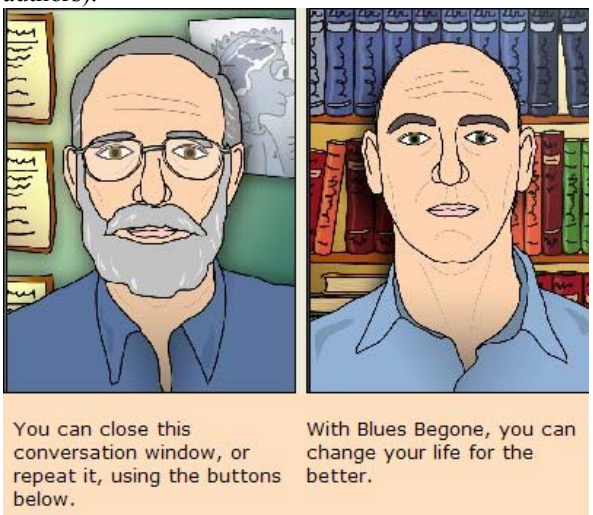


Figure 4 Conversation Heads

6.5 Cartoon

Blues Begone makes extensive use of static illustrative cartoons and animations. About 60 are included. Figure 5 shows a typical cartoon.

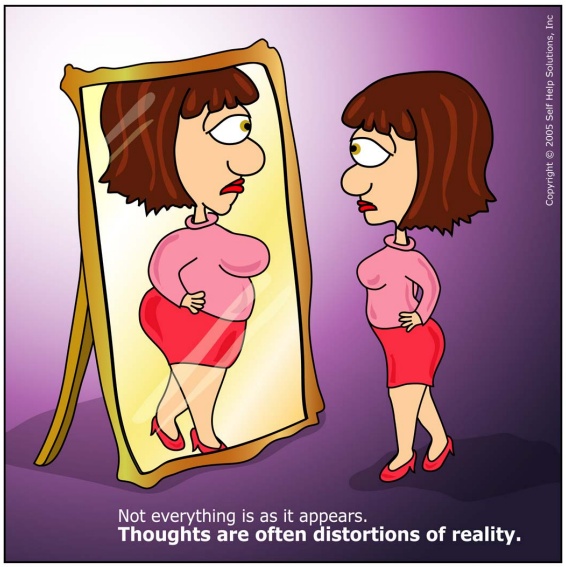


Figure 5 Sample Cartoon

6.6 Case Study

A Case Study is a collection of a small number of the roadmap object types around a common theme. They are generally used to follow the progress of an (initially) depressed person. Case Studies are also represented by XML documents. Blues Begone includes six complete Case Studies.

6.7 Additional Items

Each episode in Blues Begone includes a hint (information on how to effectively use the program), a tip (suggestion on something that is likely to improve mood) and a joke (that most users do not find funny. The challenge is to find jokes that amuse everybody, but offend nobody.)

Some aspects of the program are adjusted based on religious inclinations. A person's faith can be a strong factor in recovery.

Blues Begone frequently offers context-dependent encouragement and feedback.

A user may be assigned homework to do outside time with the computer. Blues Begone arranges a follow up date, and checks on the user's progress when the homework is due.

From time to time, Blues Begone generates surprises.

7. Packaging and Delivery

The Blues Begone installation is about 1 gigabyte, and so a little too large for direct downloading over

the public Internet. The Blues Begone Personal Edition consists of a 2 CD set: one CD for software installation, the other a resource CD that must be present for the software to function. An institutional version allows a large number of users to work on a single computer. A corporate version allows unlimited downloads and installations from an intranet site, and does not require a resource CD to be present for use.

8. Validation

Blues Begone has the intended purpose of helping people recover from depression in a private, anonymous situation. The corresponding validation questions are:

- Do users in fact show a reduction in depression and anxiety?
- Will users actually apply themselves to using Blues Begone without the personal contact of a human therapist?

We do not yet have statistical data to answer the first question, and have only qualitative data from independent interviews of users for the second. An interesting picture emerges from the qualitative interviews. Users describe a "virtual" relationship they develop with Blues Begone and its animated heads.

An interesting tension begins to emerge depending on how we view the therapeutic process. If we think of the therapeutic tool like a drug, and apply its life cycle, then the sequence is; develop, test through rigorous trial, obtain approval, then freeze. By its nature this life cycle is opposed to the concept of continuous quality improvement.

9. Future Work

Some clinical trials are presently under way with depressed users. These trials will yield efficacy data.

Computing technology is always marching on. A product that works satisfactorily now is likely to have problems with future updates of operating systems, hardware and user expectation. We plan to update the design to make it a genuinely generic self-help engine having a highly data driven therapeutic content.

10. Concluding Comments

Blues Begone is an innovation in delivering self-help to people suffering from depression and anxiety.

11. References

[1] "ELIZA - A Computer Program for the Study of Natural Language Communication between Man and Machine," *Communications of the Association for Computing Machinery 9* (1966): 36-45.

[2] Freud, S., A *General Introduction to Psycho-analysis*. First lecture states: "… when we undertake to treat a neurotic … explain to him … its long duration, the trials and sacrifices which will be required of him … and … we can make no definite promises …."

[3] Clarkson, P., *The Therapeutic Relationship*, Whirr, London, 1995. Page 4 states: "More and more research studies … demonstrate that it is the relationship between the client and the psychotherapist, more than any other factor, which determines the effectiveness of psychotherapy."

[4] Smith, N. M., M. R. Floyd, C. Jamison, & F. Scogin, "Three-year follow-up of bibliotherapy for depression. *Journal of Consulting and Clinical Psychology*, 65(2), 324-327.

[5] Beck, A. T., *Depression Causes and Treatment*, University of Pennsylvania Press, Philadelphia, 1967

[6] Clinical Guideline 23, "Management of depression in primary and secondary care" National Institute for Clinical Excellence, December 2004

[7] See www.ultrasis.com

[8] Technology Appraisal 97, "Computerised cognitive behaviour therapy for depression and anxiety" National Institute for Clinical Excellence. February 2006

[9] See www.ccbt.co.uk

[10] See www.calipso.co.uk

[11] The World Health Report 2001, World Health Organization. Geneva

[12] Robins LN, Regier DA (Eds). *Psychiatric Disorders in America, The Epidemiologic Catchment Area Study,* 1990; New York: The Free Press.

[13] American Psychiatric Association. (1994). *Diagnostic and statistical manual of mental disorders* (4th Edition), Washington DC. 339ff.

[14] Beck, A. T., R. Steer & M. Garbin, "Psychometric Properties of the Beck Depression Inventory: Twenty-five years of evaluation." *Clinical Psychology Review*, 8 (1988),77-100.

[15] Radloff, L. S., "The CES-D Scale: A self report depression scale for research in the general population." *Applied Psychological Measurement*, 1, (1977), 385-401

[16] Microsoft Speech Application Programming Interface. See http://www.microsoft.com/speech/download/sdk51/

A Hybrid View in a Laparoscopic Surgery Training System

Chuan Feng, Jerzy W. Rozenblit, PhD
Electrical and Computer Engineering Department
The University of Arizona
Tucson, Arizona 85721
Email: {fengc, jr}@ece.arizona.edu

Allan J. Hamilton, MD
Arizona Simulation Technology
and Education Center
The University of Arizona
Tucson, Arizona 85721
Email: allan@surgery.arizona.edu

Abstract

In this paper, a hybrid view application is proposed — a subsystem of a computerized laparoscopic surgery training system. To minimize the potential hazards of laparoscopic surgery, an assistive training system is being developed. A digital camera and magnetic position sensors are used to detect laparoscopic instruments in the system. The hybrid view is a component of this system which overlays the positions of organs and objects with the path history of the instruments. This method could help confirm erroneous movements made by surgeons and provide more useful information than separate sensors. This may minimize the cognitive overload on the surgeons. Initial experimental results are presented to show the feasibility of the proposed method.

1 Introduction

Laparoscopic surgery is performed with an endoscope and several long, thin instruments through small incisions. Due to its minimally invasive nature, laparoscopic surgery offers advantages such as shorter recovery time and reduced pain compared with traditional methods. However, inexperienced surgeons often lack a correct perception of an instrument's position due to a restricted vision field, hand-eye coordination problems, limited work space and the lack of tactile sensation. Those issues make laparoscopic surgery a difficult skill for medical students and residents to master.

There has been some research on the effectiveness of different kinds of training and guidance. Traditional surgical training methods such as using animals and cadavers have limitations because animals do not have the same anatomy as a human being and cadavers can not provide correct physiological responses. Surgery simulation is increasingly perceived as a valuable addition to traditional medical training. According to [3], laparoscopic training translates into approximately 30-35% more efficiency as measured by operative time and decreased complication rates compared to a control group not receiving simulation training.

One representative simulation tool-set is called the Pelvi-trainer [4]. It is a box that simulates the abdomen, with apertures for the insertion of instruments and camera. Trainees use real instruments to practice basic skills by manipulating objects or interacting with artificial tissue and anatomical models, using a video display for visualization. The main limitation of this approach is the absence of objective performance assessment, a feature available in some Virtual Reality (VR) systems, which use computer to simulate the whole training procedure. Hamilton [5] reports that speed was the only end-point measured within the pelvi-trainer while the virtual reality (VR) simulator reports error for each task performance [6]. Limitation of the VR simulations include inadequate realism of the virtual environment, inaccurate haptic feedback and the exorbitantly high cost of these systems.

Our vision for the proposed Virtual Assistant Surgical Training (VAST) system [1] is to bridge the gap between pelvi-trainers and VR systems, combining the advantages of both approaches to design a system that is simple and effective. We propose a knowledge-based sensor system to provide training prior to surgery and assistance during surgery. Our design features the embedding of micro-sensors into the instruments employed for simulation training. The detection and recording of instrument movement permits our system not only to measure a trainee's progress in acquiring psychomotor skills and compare these data to normative databases, but also to evaluate instrument effectiveness in reducing error. From a training perspective, the sensor-based system tracks and returns information on various performance metrics such as position and velocity of instruments, total path length of motion, erratic movements, time taken, number of attempts, dexterity, etc.

Fig. 1 contrasts the Knowledge-based VAST System with the traditional approach. In the VAST system, the sur-

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

geon acts upon the patient or simulator through instruments and receives visual and force feedback from the supervisor both in the operating room and training settings. The supervisor represents the sensing interface and the knowledge-based computer system. It consists of a sensor fusion engine at the front-end and a knowledge based inference system at the back-end. In this paper, we discuss the issues related to the sensors.

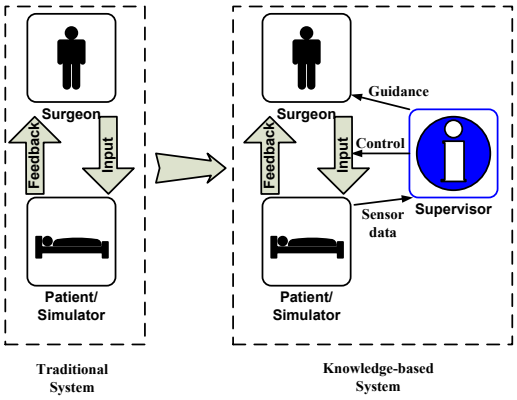


Figure 1. VAST system

A prototype system is being developed which is capable of high fidelity motion tracking of surgical instruments and basic performance assessment analysis. To our knowledge, this is the first approach that uses different kinds of sensors to assist in accurate tracking of instruments' positions and movements. The system fuses data from sensors and provides information to surgeons. To enable the data to be correctly understood, the various sensor outputs must be fused to provide a single representation.

Two kinds of sensors are implemented within the training system, a CCD camera and magnetic kinematics sensors. The Hybrid View, shown in the Fig. 2, which overlays the positions of organs and objects with the path history of the instruments, is one of the components of the VAST system. Immediate visual feedback obtained from the hybrid view helps surgeons check and evaluate the entire training process to minimize potential hazards, identify problem areas and find solutions.

For successful surgery, the laparoscopic camera should be moved from time to time to adjust the view of the surgical site. How to maintain data integrity during camera movement is the key issue of the hybrid view. Some researchers track the instruments' position using image analysis [2]. However, reliability is questionable because image analysis is highly related to the lighting conditions and attitude of the instruments. This can also make the tracking results discontinuous. For example, if the instrument is blocked by tissue or blood, vision tracking is unavailable. On the other hand, magnetic sensors can provide continuously accurate 3D position information on the instrument but lack the mapping relationship between the 3D space and 2D image. In order to solve the issue without more specialized hardware, a multi-level processing method was used.

In the remainder of the paper, we discuss our proposed approach in more detail. In section 2, the hybrid view generation system is presented in detail. In Section 3, several experiments are described must prove the feasibility of the proposed method. Section 4 is the conclusion.

2 System Design

2.1 System Architecture

In our application, shown in Fig. 3, a system with three processing levels generates the hybrid view.

The sources of information which include different kinds of sensors and knowledge database, are indicated on the left side of Fig. 3. A CCD camera and $microBIRD^{®}$ 6-DOF magnetic kinematics sensor are used in our application [7]. The CCD camera is connected to an endoscope which provides live image of the operating site. The kinematics sensing system includes a magnetic field transmitter, two position sensors (1.3mm in diameter) which can be mounted on the tip of the laparoscopic instruments and a PCI interface data processing card. The transmitter remains fixed to pro-

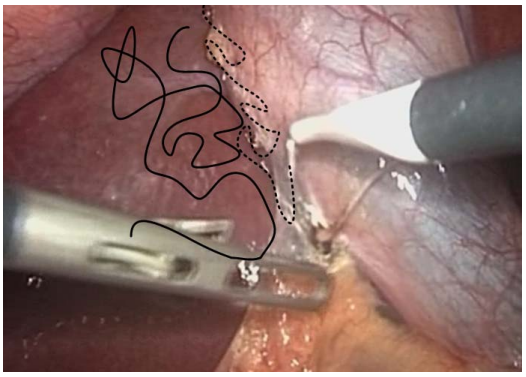


Figure 2. Hybrid View Sketch Map

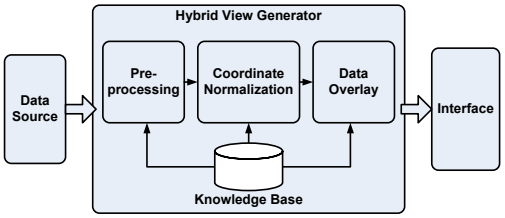


Figure 3. Hybrid View Generating System Framework

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

vide a Cartesian frame of reference for position tracking.

The right side of the figure shows the system interface. The interface consists of a subsystem interface and a human computer interface (HCI). The subsystem interface exchanges information with the VAST system and the HCI allows a human operator to input commands and get information from the generator. A *hybrid view* will be output through the HCI so that it can be watched by surgeons. The hybrid data will also be sent to the inference module of VAST through the subsystem interface. Therefore, further evaluation and feedback can be made. In a training setting, the trainee is required to repeat the motion within the safety bounds until a motion rule check passes. This enforced learning process should help the trainee master the necessary skills.

Three processing levels are shown in the system architecture. The first level is Source Preprocessing, which synchronizes the information flow from different sensors and distributes data to appropriate processes. The second processing level is Coordinate Normalization, which transforms sensor data into a consistent set of coordinates and estimates the position and kinematics of the instruments. In the third processing level, Data Overlay, data are fused together so the hybrid view can be generated. Detailed design information is provided below.

2.2 Level 1: Source Preprocessing

Source Preprocessing is the first level of the hierarchical processing model, which guarantees the synchronization of different data sources. Because the overall objective is generating the correct instrument track over the operating site, the camera drives this process. When a new frame is captured by the camera, the Source Preprocessing level forces the microBIRD sensor to obtain the corresponding 3D coordinates of the instruments and to distribute the data to the next processing levels.

Image filters are applied for noise elimination. A median filter is used to minimize the noise because it reduces the effect of unrepresentative pixels with almost no degradation to the underlying image.

Color segmentation method is used to finish the image preprocessing. Before color segmentation, a color cluster training process [11] is necessary. First, a color that does not appear within typical laparoscopic images is chosen to mark the instrument. Then an image of the color-marked instrument will be taken under real working light conditions. The mark on the instrument can then be outlined manually to provide a color cluster of the mark within the RGB color space.

During the working period, all pixels within the image are distinguished by color segmentation. Each pixel whose RGB value falls within the pre-training color cluster will be treated as one element of instrument area set I. Therefore, the color image can be converted into a binary data matrix which distinguishes the exact pixels of instruments. Each pixel in the image is mapped to the correspondent position of the data matrix.

2.3 Level 2: Coordinate Normalization

If there are no abnormal signals from the Source Preprocessing level, level 2 — Coordinate Normalization will do further processing. This level transforms sensor data into a consistent set of coordinates and estimates the position and kinematics of the instruments.

Before an operation, a rule module is applied to infer the information sent through the previous processing level. Sample rules are shown here:

- *if* microBIRD sensor data are incorrect
  then refinement to kinematics sensor data is disabled.
- *if* image data are incorrect
  then refinement to camera data is disabled.
- *if* both kinds of sensor data are incorrect
  then cancel the current working cycle.

The purpose of the module is to synchronize the functional modules according to the source data. Only the validated sensor data can be used for further processing.

2.3.1 Coordinate transformation of the kinematics sensors

The intent of the data fusion system is to generate a consistent Hybrid View, so transforming the sensory data into the image coordinates is an necessary procedure. Camera calibration is the first step when trying to map 3D information to a 2D image taken by a camera.

In order to provide real time computing ability, a simple linear mapping algorithm called Direct Linear Transformation (DLT) is implemented, which uses a linear camera model to extract pixel coordinates from the 3D data. The DLT mapping function can be expressed as equation 1:

$$\begin{aligned} u &= \frac{L_1 X_w + L_2 Y_w + L_3 Z_w + L_4}{L_9 X_w + L_{10} Y_w + L_{11} Z_w + 1} \\ v &= \frac{L_5 X_w + L_6 Y_w + L_7 Z_w + L_8}{L_9 X_w + L_{10} Y_w + L_{11} Z_w + 1} \end{aligned} \tag{1}$$

Where $L_1 \ldots L_{11}$ are the parameters of transformation, which are the combination of the camera is intrinsic and extrinsic parameters. (X_w, Y_w, Z_w) is the 3D space coordinate, and the (u, v) is 2D image pixel. This equation can be solved by the least-square algorithm [8].

The drawback of DLT is that the calibration method cannot adapt to camera movements during the operation. To

solve this issue, we need to modify the camera parameters in real time.

Because intrinsic parameters will not change when the camera moves, our approach was to modify the extrinsic parameters only. A virtual camera algorithm is applied as follows:

Assume the camera is fixed in a specific position, after the calibration. An isomorphic mapping function can be built as follows: 2:

$$\lambda \begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = f(X_c) = f(\begin{pmatrix} x_c \\ y_c \\ z_c \\ 1 \end{pmatrix}) \tag{2}$$

Where u, v are the pixel coordinates in image plane and x_c, y_c, z_c are the 3d point coordinates in world system, which is called S_c. X_c is the expended vector $(x_c, y_c, z_c, 1)^T$.

When the camera moves, the world system S_c will not move, so there exists a transform matrix M, which indicates the position and orientation of the camera movement. The mapping function is:

$$X'_c = M \cdot X_c \tag{3}$$

The original mapping function in 2 can be used. However, the point X_c has been transformed into a new coordinate X'_c within the new world system. So the point X'_c will be mapped to a new pixel $P' = (u', v')$ as below:

$$\lambda \begin{pmatrix} u' \\ v' \\ 1 \end{pmatrix} = f(X'_c) = f(M \cdot X_c) \tag{4}$$

From equation 4, we can confirm that the camera calibration parameters L need not be modified any more. We only need to continuously refresh the transformation matrix M so that it can express the transform relationship. In the next processing level, a detailed solution for refreshing the transformation matrix is described, which relates to image analysis.

2.3.2 Coordinate transformation of the camera data

The image data matrix sent from the Source Preprocessing level is a 2D binary data set that indicates the result of color segmentation. The position of the instrument must be acquired from the image data. The center of mass is used to indicate the instrument's position. Within the binary image, the center of mass can be calculated according to equation 5:

$$P = \frac{1}{M} \sum m_i r_i \tag{5}$$

Where M is the sum of the particle pixels; r_i is the position of the ith pixel, which is the distance from the pixel to the origin of the image; m_i is the value of the ith pixel; P is the center of mass.

2.4 Level 3: Data Overlay

The purpose of the Data Overlay level is to generate the hybrid view from different data sources.

As we previously mentioned, the transformation matrix M is used to indicate the shift and rotation of the camera. M is set to a unit matrix when the system is initialized since there is no camera movement. When the camera is moved according to surgical need, the matrix must be refreshed in real time.

If the 3D coordinates and the corresponding 2D pixels are known, the only unknown of the equations e:transform9 is the M, which consists of 6 moving parameters of the camera: $(x, y, z, \theta_{pitch}, \theta_{yaw}, \theta_{roll})$. These points, with known 3D to 2D mapping relationship are called reference points. Thus, when the camera moves, the first three reference points can be used to generate equations of M. The Newton method is used to solve the nonlinear equations. It is easily implemented in the system without any significant computational power consumption.

According to the feature of DLT, the matrix obtained from the above algorithm can only fit a small area close to the current position of the instruments. Therefore, the matrix needs to be modified continuously even when the camera is fixed. In order to efficiently modify the transformation matrix, the error E is defined as below:

$$E := (\Delta u, \Delta v) = (u - u', v - v') \tag{6}$$

Where (u, v) is the pixel position got from image analysis and (u', v') is the pixel got from 3D to 2D mapping. E is the vector indicating the difference between 2 separate data sources. The modification rule is indicated in formula 7.

$$\begin{array}{cc} if & \sum |E| > T_E \\ then & refresh \\ else & continue \end{array} \tag{7}$$

Where T_E is the threshold of accumulated error, and $sumE$ is the integration of errors, which can eliminate random noise and calculation inaccuracy.

After obtaining the transformation matrix of the 3D to 2D mapping, the hybrid view is generated. At the beginning, information sent from the lower processing level is analyzed by the knowledge based engine. If no data are available, the current fusion cycle will be ignored and nothing will be generated. If only one data source is verified, the tracks will be generated according to the one source. Otherwise, the camera data is used to draw the hybrid view; at the same time, data from both sources are used to refresh the 3D to 2D transformation matrix. The flow chart is shown in Fig. 4.

3 Experiment

To evaluate the proposed method, several experiments have been conducted. We will describe the camera calibration, image analysis and the 3D-2D transformation results separately in this section.

3.1 Camera calibration

Camera calibration is the first step of the hybrid view fusion process. In our application, DLT is used to obtain the camera parameters. To acquire the camera parameter vector L, a calibration testing board is applied. The calibration board contains a rectangular coordinate network of 1×1 inch in size with one inch margins. Thus the 3D coordinates in the world space and the corresponding 2D pixels can be determined easily.

The experiment process is described below:

- Fix the camera and calibration board in the space.
- Capture a picture of the calibration board by the camera.
- Determine $n(n > 6)$ pixels, as shown in Fig 5.
- Determine the related 3D points coordinates.
- Calculate the calibration parameters L.

Table. 1 shows the calibration error. The accuracy is high enough and sufficient for the hybrid view application.

3.2 Image analysis

The image analysis experiment is shown in Fig. 6:

The left picture is the image captured from the camera. Our aim is to determine the position of the color mark area.

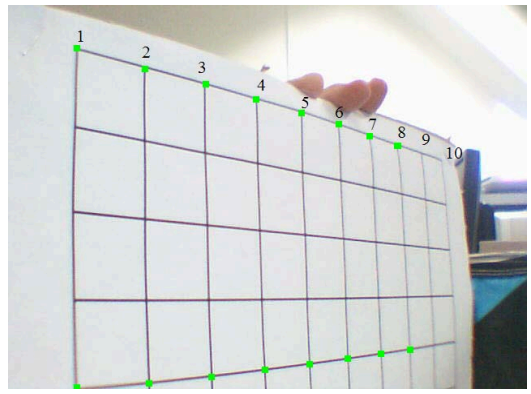


Figure 5. Camera Calibration Board

Table 1. Camera calibration results

3D coordinate (mm)	2D (pixel)	3D-2D map (pixel)	error (pixel)
(7.18,-10.16,7.18)	(495,430)	(496.4,431.2)	1.8411
(5.39,-20.32,5.39)	(447,162)	(446.7,161.7)	0.4036
(3.59,-10.16,3.59)	(419,441)	(417.9,441.2)	1.1608
(0.00,-20.32,0.00)	(306,117)	(305.3,118.5)	1.6611
(-1.80,-10.16,-1.80)	(251,464)	(251.0,462.4)	1.6467
(-3.59,-20.32,-3.59)	(168,77)	(170.1,77.2)	2.0993
(-5.39,-10.16,-5.39)	(84,479)	(84.0,483.5)	4.5198

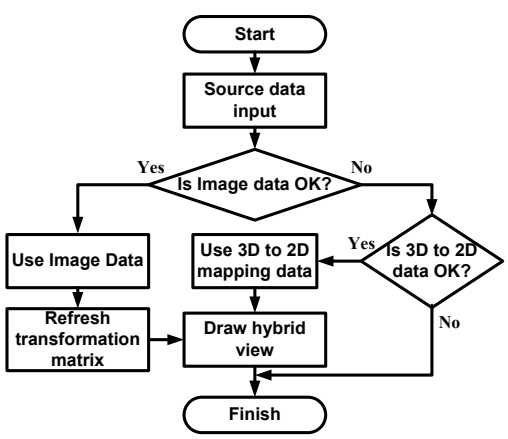


Figure 4. Hybrid view generation flow chart

Figure 6. Image Analysis

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

The middle picture shows the binary image after color segmentation (the red color mark has been segmented well). The right picture is the result of mass point calculation. The green cross indicates the mass point of the color mark.

3.3 3D to 2D transformation

3D-2D transformation is the most complex part within the system. It is also the most important part. The calibration board is used to show the feasibility of the algorithm described above. The experiment is set up as follows:

- Capture a picture at position 1.
- Calculate camera parameters L according to the reference points.
- Move camera to position 2 and capture a new picture.
- Calculate a transfer matrix using 3 reference points.
- Calculate pixels from 3D space according to camera parameters and transferring matrix.

The diagram in Fig. 7 shows the experiment process. Two pictures captured by the same camera in two different positions are shown. The geometric relations between the camera and calibration board are shown at the bottom of the picture. The camera takes the first picture at position 1, then moves to position 2. During the experiment, the calibration board is fixed, so the 3D coordinates of the reference points will not change. Three reference points are chosen in the second picture to calculate the transformation matrix. After the calculation, a virtual path in the left picture is mapped to the right picture. Although they are the same points in 3D space, due to the movement of the camera, the hybrid paths are different. This is the reason we use the hybrid view, which correctly describes the 3D to 2D mapping relationships.

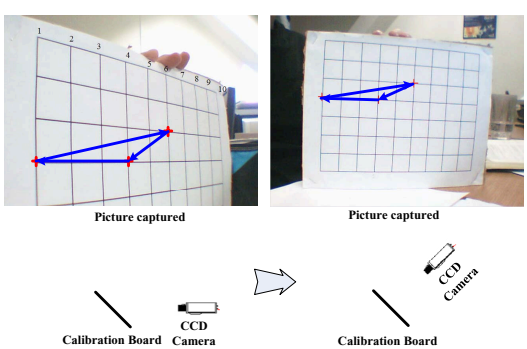


Figure 7. 3d to 2D mapping experiment

After mapping the 3D space coordinates onto the 2D image pixels, some errors can be found, as shown in Fig. 8. The reasons for the errors are twofold: one is the calibration method itself, the other is the transfer matrix. We use a linear transformation method to describe the mapping function which discards the nonlinear distortion of the lens, thus errors are unavoidable. The transformation matrix is acquired by the Newton method, which is a nonlinear optimization method, not an exact solution. According to the error diagram in Fig. 9, the points near the reference points are more accurate than the points more distant. Therefore, we need to refresh the 3D-2D transformation matrix frequently.

In the VAST system, the microBIRD sensor is used to in place of the calibration board for acquiring the 3D space coordinates. All the other operations are the same as the experiments described above.

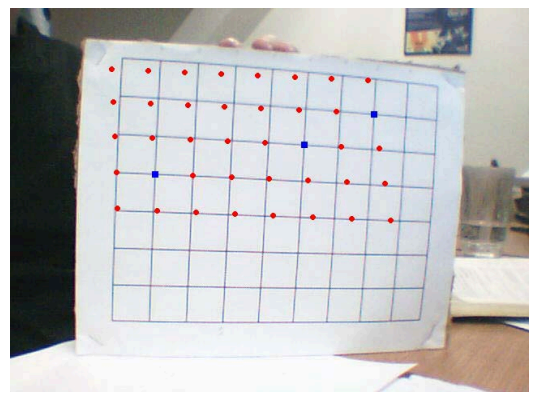


Figure 8. Pixels obtained from transformation

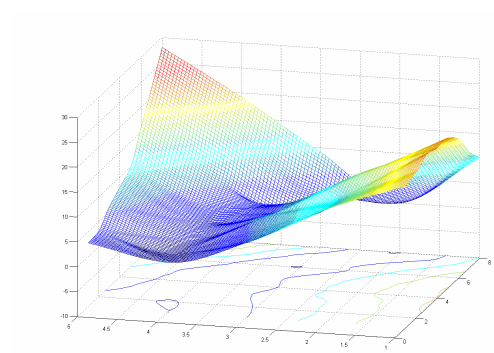


Figure 9. Error Diagram of transformation

4 Conclusion

In this paper, a novel method of the realizing a hybrid view within a virtual, assistive surgical training system has been presented. The hybrid view, which is a subsystem of a computerized laparoscopic surgery training system, helps

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

confirm the erroneous movements without more expensive sensors while reducing the complexity of the system. In the proposed method, we use a multi-level processing model to describe the architecture of the system. Intensity information of the image analysis, 3D space Euclidean transformation, and digital camera calibration methods have been presented. Experimental results demonstrate the feasibility of the proposed methods. Further advances need to be made to provide three dimensional vision through a stereo endoscope and haptic information from the tip of the instruments. Currently, we are testing the system with medical students and residents.

References

[1] C. Feng, H. Haniffa and et al., "Surgical Training and Performance Assessment Using Motion Tracking," *EMSS 06*, 2006.

[2] G. Wei, K. Arbter and G. Hirzinger,"Real-time visual servoing for laparoscopic surgery. Controlling robot motion with color image segmentation," *IEEE Engineering in Medicine and Biology Magazine*, Vol. 16, Issue. 1, pp. 40-45, 1997.

[3] J. Korndorffer, Jr, J. Dunne, and et al., "Simulator Training for Laparoscopic Suturing Using Performance Goals Translates to the Operating Room," *Journal of the American College of Surgeons*, Vol. 201, Issue. 1, pp. 23-29, 2005.

[4] S. Shrivastava, R. Sudarshan, and et al., "Surgical Training and Performance Evaluation using Virtual Reality based Simulator," *Virtual Concept 2003, France*, 2003.

[5] E.C. Hamilton, D.J. Scott and et al., "Comparison of video trainer and virtual reality training systems on acquisition of laparoscopic skills Journal Surgical Endoscopy," *Surgical Endoscopy Journal*, Vol. 16, pp. 406 - 411, 2002.

[6] Simbionix Ltd., *http://www.simbionix.com/*, 2006.

[7] Ascension Technology Co., "microBIRD Technical Reference Guide," *http://www.ascension-tech.com/* , 2005.

[8] Y.I. Abdel-Aziz, and H.M. Karara, (1971). "Direct linear transformation from comparator coordinates into object space coordinates in close-range photogrammetry," *Proceedings of the Symposium on Close-Range Photogrammetry*, pp. 1-18, 1971.

[9] J. Leonard, "Sensor Fusion for Surgical Applications," *The 15th AESS/IEEE Dayton Section Symposium*, pp.37-44, 1998.

[10] L. Joskowicz and R.H. Taylor, "Computers in imaging and guided surgery," *IEEE Computational Science and Engineering*, pp.65-72, 2001.

[11] X. Zhang and S. Payandeh, "Application of Visual Tracking for Robot-Assisted Laparoscopic Surgery," *Journal of Robotic Systems*, pp.315-328, 2002

[12] B. Law, M.S. Atkins and etal. "Eye gaze patterns differentiate novice and experts in a virtual laparoscopic surgery training environment," *Proceedings of the 2004 symposium on Eye tracking research and applications*, pp. 41-48, 2004.

[13] F. Tendick and M.C. Cavusoglu, "Human-Machine Interaces for Minimally Invasive Surgery," *19th IEEE EMBS*, pp. 2771-2776, 1997.

[14] J. Llinas and D.L. Hall, " An introduction to multisensor data fusion," *Procedings of the IEEE*, pp.6-23, 1998.

Modeling the Functionality of Multi-Functional Software Systems*

Alexander Gruler, Alexander Harhurin, Judith Hartmann
Technische Universität München
Department of Informatics
Chair of Software and Systems Engineering
Boltzmannstr. 3, 85748 Garching, Germany
{gruler,harhurin,hartmanj}@in.tum.de

Abstract

Today, many software-based, reactive systems offer a multitude of functionality. One way to master the development of such a system is to model its functionality on an abstract level and derive a system architecture and an implementation out of this functionality model. In this paper, we present an approach to model the functionality by means of related, interacting services. For us, a service represents an single functionality of a system. The concept of services is used in two consecutive model layers with well-defined semantics leading from a black-box description of the system to a white-box model which consists of communicating services. Due to the precise semantics of a service and the interaction of services, the service models can be directly refined to a logical component architecture which in turn integrates into the development of a concrete implementation of the overall system.

1 Introduction

Today, many software-intensive systems provide a wide range of functionality, i.e. they offer a variety of different, user-observable functions. We call such systems *multi-functional*. For us, a *multi-functional system* is any reactive system which offers a set of different functions and combines them into a single system in consideration of their mutual dependencies. Thereby, the functionality of the overall system exhibits a surplus value compared to the set of individual functions. With *functionality* we mean the characteristic, observable behavior of a system, or more precisely its reaction (outputs) to certain inputs.

The property of multi-functionality spans various application domains such as telecommunication, avionics or automotive. Here for example, the functionality offered by an automobile has been increasing rapidly during the last decades, having reached a state where a modern premium class automobile has advanced to be a highly versatile multi-functional system. Today, the system functionality is mainly realized by software. Certainly, the efficient development of software for such complex, multi-functional systems requires special techniques and methods.

Addressing this trend, we introduce an approach to model a multi-functional system during the early phases of a model-based development process by means of two integrated *service models*. The service models capture the pure functionality of the system in a formal way and are the basis for further, more detailed architecture models, such as e.g. a logical component architecture. In particular, they establish a formal relation between functional requirements and architecture models. In both service models the concept of a *service* is used to independently model single functionalities of the system which are related and combined to form the overall system behavior.

We call the upper, more abstract model the *Service Diagram*. It gives a structured view of the overall functionality offered by the system as a hierarchy of all services directly observable by the user/environment. Subsequently, the Service Diagram is refined into a consecutive, less abstract model, the so-called *Service Network*. It gives a more detailed view on the system, by now considering the interaction between the identified services. Together with the Service Diagram, the Service Network provides the basis for the construction of the *Logical Architecture*.

The rest of this paper is organized as follows: In Section 2 we briefly introduce an example of a multi-functional system, which will be used throughout the rest of the paper to illustrate the suggested concepts. Section 3 presents current issues in the modeling of multi-functional systems and motivates the presented techniques. In Section 4 we introduce

*This work was partially funded by the German Federal Ministry of Education and Research (BMBF) in the framework of the VEIA project under grant 01ISF15A. The responsibility for this article lies with the authors.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

and briefly define the basic concepts such as the idea of a *service* and motivate their relevance for the use as means of describing the functionality of multi-functional systems. Section 5 can be seen as the core of the paper, since here, we describe how the basic concepts should be applied in order to formally capture the functionality of a system. In particular, we describe the two service models as well as the model of the logical components. Finally, we compare our models with related approaches in Section 6 before we conclude the paper in Section 7 together with an outline how to integrate the concepts in our future work.

2 Running Example

The introduced techniques and models will be illustrated by the example of a door control unit (DCU) [13]. Since in a modern premium class car the whole functionality offered by the DCU is exclusively software based, it gives a realistic example of a multi-functional system with clear distinguishable sub-functionalities. In the following, we briefly describe the functionality of the DCU.

The DCU controls several comfort features of a car, such as adjustment of the power front seats, memory functionality, seat heating, door lock, power windows, interior lighting, and the adjustment of the outside mirrors.

The DCU provides a physical user interface consisting of several buttons and switches mainly located in the front and back door lining. The functionalities are as expected, e.g. adjusting the seat in its horizontal and vertical axis, changing the angle of the seat back and the extension of the head restraint, opening and closing the windows, saving the positions of the seat, mirrors and steering wheel, turning on the seat heating and changing its degree of intensity, and adjusting the vertical and horizontal angle of the outside mirrors and turning on/off the mirror heating.

Dependencies between different functions and other relevant details will be described at the appropriate places.

3 Contributions

In general, the software engineering process and the respective methods for the development of complex, multi-functional systems has not reached a stage yet which satisfies the current needs of the industry. This makes the development of such systems a challenge which requires special techniques and methods. With the models and concepts described in this paper we address the following issues concerning the development of complex multi-functional systems.

During the early phases of a model-based development process, i.e. during the transition from requirements to architecture models, an open issue is at what level to start with a formal description. In practice today, functional requirements are not precisely formulated. The usual approaches to modeling requirements or the functionality offered by a system are use case diagrams [14] or feature models [15] which both lack a precise semantics [8] in general.

In contrast to a pure informal approach, we introduce a formal model with a well-defined semantics for describing the functionality already at an early stage of the development process. This has several advantages: Firstly, a formal model which formalizes (functional) requirements allows an automatic analysis of the system already in the early phases of the development process. By this, discrepancies between conflicting functionalities can be detected and resolved. Secondly, a formal model can be simulated – in the case of the service models with the CASE-tool AutoFocus [21] – which is a valuable property for industrial application.

The increasing complexity of multi-functional systems requires to design a system in a modular fashion by splitting up the system into an appropriate set of different sub-functions and generating the overall system as a combination of these. This implies to model sub-functionalities independently and to compose/combine them adequately afterwards in order to form the overall system behavior. In both our models we realize a modular, independent specification of (sub-) functions by means of modular *services*. A single service can consist of several sub-services with the meaning that a service aggregates the functionalities expressed by its sub-services. On the one hand, this allows a great freedom in the specification of an individual service, but on the other, it results in a structured hierarchy of services which forms a single service representing the overall functionality of the system.

Concerning the development process a system will be modeled at different levels of abstraction in a way that each level gives a more or less abstract view of the system. The fact that both models are based on the same notion of a service facilitates the transition from the Service Diagram to the Service Network. Thus, both service models integrate seamlessly at the top of such a model chain closing the formal gap in a model-based development process. In particular, they provide the basis for a formal transition from (functional) requirements to architecture design which currently is not well supported by formalisms.

The abstraction level where a system is seen only according to its functionality is an appropriate place to realize changes due to an evolution of the system. Every functionality modularly modeled here can be traced to a set of implementation entities (such as components) in subsequent models, which means that changes in the service models can easily be propagated in subsequent models. Thus, the service models represent a suitable abstraction level with a high re-use potential.

4 Service Theory

Before we describe the service layers in detail in Section 5 we introduce the necessary basics in this section. The following definitions and ideas are based on the Service Theory [3] introduced by Broy which itself is based on the FOCUS theory [5] for the specification of interactive systems. Since the FOCUS theory assigns a precise semantics to each of its concepts, it is a suitable basis for the ideas introduced in this paper.

We use the concept of services to capture, structure and relate the functionality offered by a system. So far, this is similar to *feature*-based approaches (see Section 6), but in contrast to a feature a service has a precise semantics defined by its input/output behavior.

Basically, a service is based on the idea of timed data streams which are used to model the interaction of a service by describing the communication with its environment. Intuitively, a *timed (data) stream* can be thought of as a chronologically ordered (finite or infinite) sequence of data messages. Given a set M of data messages, we denote a timed stream of elements from M by a function $s : \mathbb{N} \rightarrow M$. We assume a model of time consisting of an infinite sequence of time intervals of equal length. In each time interval only one message can be transmitted. Such streams can be used to represent histories of communications of data messages transmitted within a time frame. For each time interval $t \in \mathbb{N}$, $s(t)$ denotes the message communicated within the time interval t.

Every service provides a syntactic service interface and a behavioral semantics. The *syntactic interface* of a service is given by the set of all typed ports of the service. We write $I \blacktriangleright O$ to denote the service interface, where I is the set of input ports and O the set of output ports respectively. With every port we associate a stream representing the message history of this port. Given a service S with syntactic interface $I \blacktriangleright O$ for each port $p \in I \cup O$ and all time intervals $t \in \mathbb{N}$, the term $S[p](t)$ denotes the message communicated via the port p within the time interval t. Note that a service can interact with its environment exclusively via its ports.

The *behavioral semantics* of a service S with syntactic interface $I \blacktriangleright O$ is precisely characterized by a partially defined (stream-processing) function mapping streams of messages received on the input ports $q \in I$ to streams of corresponding messages on the output ports $p \in O$. With "partially defined" we mean that a service does not always have to return a well-defined output, i.e. the stream-processing function, which characterizes the behavior of the service, does not have to be defined for all possible inputs. With $Dom(S)$ we denote the set of input streams of the service S which have defined outputs.

Services can be connected through *channels*. The idea is that a directed channel (p, q) between two ports p and q represents a connection between two services S_1 and S_2 with compatible[1] service interfaces $I_1 \blacktriangleright O_1$ and $I_2 \blacktriangleright O_2$, where $p \in O_1, q \in I_2$ respectively.

All in all, a service is an appropriate concept to describe functionalities offered by a multi-functional, reactive system during early stages of the development where the focus of the developer is to model all information that is already known about the system while not bothering about the situations which are not important for the current level of abstraction. In the next section we will see how these concepts can be used in order to describe a system as a set of related services.

5 Functionality Model Layers

In this section we introduce a framework of layers which gives different views on the system from different levels of abstraction respectively (see Figure 1). This corresponds to the idea to specify a system on consecutive abstraction layers, each one giving a more detailed model of the system, where the highest layer reflects a very abstract, informal description of the systems, while the lowest layer represents a concrete deployable implementation.

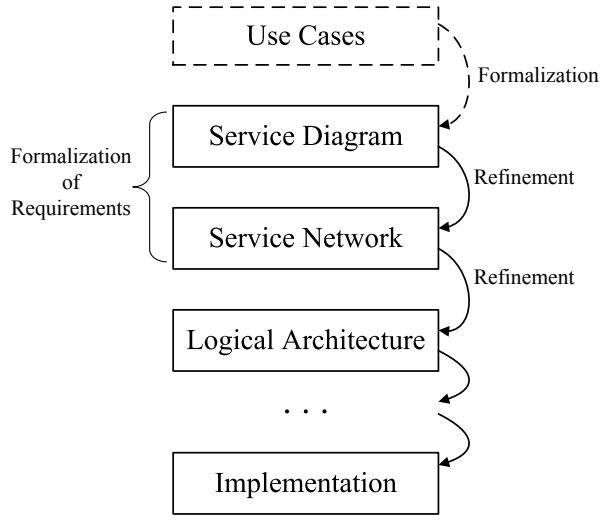


Figure 1. Layer Framework

Our layer framework starts from a very abstract description of the system as a set of use-cases or feature models without a well-defined semantics.

The formalization of use cases or features by independent services and their structuring yield the next layer, called *Service Diagram*. In this model, the system behavior is specified from the environment point of view (black-box

[1] For the complete definition of a connection between services see [3].

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

view). Therefore, each functionality is described by a service which is directly observable by the environment, i.e. its inputs can directly be triggered from the environment and its outputs are directly observable from the environment. Subsequently, observable inter-service dependencies are specified. Note that here, we do not characterize the communication between individual services – we only specify dependencies between them as being observable from its overall system boundaries.

Refining the Service Diagram by adding communication behavior yields the consecutive layer, called *Service Network*. It gives a more detailed view on the system, by now considering the interaction of the identified services and focusing on their intercommunication. This results in a network of communicating services which realize the functionality modeled in the Service Diagram.

The last step of our approach is to build up a *Logical Architecture* formed by a network of components, which are connected by channels, and to refine the functionality specified in the Service Network.

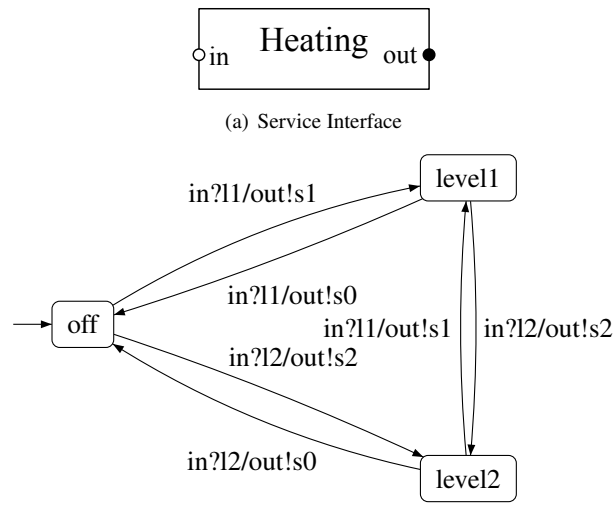
In the following, we introduce a notation for the specification of a single service, then both service models, the Service Diagram and the Service Network, are discussed in detail. Finally, we give an outline how to use these models for the construction of a Logical Architecture.

5.1 Service Specification

There are several techniques to specify the behavior of a service. In Section 4 services are formally defined by stream-processing functions. Since a service is a set of interaction patterns which give a precise relation between inputs and outputs, we propose to use modified I/O-automata [18] to specify a single service. An I/O-automaton $\mathcal{A}$ is completely defined by its set of states S, the initial state $s_0 \in S$, and the transition relation δ. A transition is denoted by

$$(s_1 \xrightarrow{in/out} s_2) \in \delta_{\mathcal{A}}, \quad \text{for } s_1, s_2 \in S.$$

A transition can be triggered in the state s_1 if all the input ports specified in the input pattern in have received the necessary input messages (denoted by $port?message$). The transition outputs data to different output ports specified in out (denoted by $port!message$), and puts the automaton into the state s_2. In other words, expression $port?message$ (resp. $port!message$) means that the message stream on the input (output) port $port$ is extended by the message $message$. Thus, the I/O-automaton constructively defines (infinite) input and output message streams as well as the relation between them. The automaton is partial in the sense that not for every input in every state there is a defined transition $t \in \delta_{\mathcal{A}}$. Since the behavior of the service is specified from the environment viewpoint (black-box view), the automaton is not allowed to have internal transitions (labeled with an empty input sequence).



(a) Service Interface

(b) Automaton

Figure 2. Specification of Service *Heating*

For example, Figure 2 shows a possible specification of the service *Heating* from our running example. This service describes a function that controls the heating of a seat. The user can switch between three states of the heating (*off*, *level 1* and *level 2*), sending one of the two messages (*l1* or *l2*) to the service. The service sends a corresponding message (*s0*, *s1* or *s2*) to the physical device responsible for the heating. The syntactic interface ($I \blacktriangleright O$) of this service is defined by I=$\{in\}$ and O=$\{out\}$. Figure 2(a) introduces a possible graphical notation. Figure 2(b) shows a behavior specification of the service. The I/O-automaton specifies a causality property between an infinite stream on the port in of input messages $\{l1, l2\}$ and an infinite stream on the port out of output messages $\{s0, s1, s2\}$.

5.2 Service Diagram

The last section showed how to to specify an individual functionality of a system modularly and independently from other functionalities. Now we concentrate on the structuring of the functionalities of multi-functional systems and their dependencies. This results in a hierarchical structure of the system functionality where the overall functionality is decomposed in services and sub-services, with defined relations between them.

The *Service Diagram* gives a specification of the behavior of systems as observable from the environment when viewing the system as a black-box, i.e. the behavior is specified as a causal relation between input and output messages. Both, the individual services offered by a system

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

and the dependencies between them are specified, but we do not consider the architecture of the system (i.e. communication links between services). Thus, we formally specify functional requirements without any predication about implementation. In particular this implies that we consider the whole system as a single (but complex) service itself composed of several sub-services.

The Service Diagram consists of single services and three kinds of relationships between them, namely *refinement*, *aggregation*, and *dependency* (cp. Figure 3).

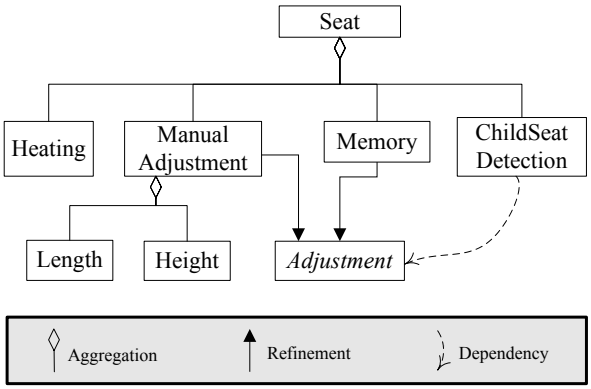


Figure 3. Service Diagram

Refinement Since a typical multi-functional system offers the plethora of functions with complex interactions between them, representing all this information without abstraction would have a negative effect on the usability of the specification. To master the complexity of a specification, we introduce *abstract services*, since the partiality of a service allows to model a certain functionality at different levels of abstraction.

An abstract service is an abstract specification of one or several functionalities. It can be considered as a contract between the services refining it and the environment – when a service refines an abstract service, it promises to provide at least the behavior specified by that abstract service. Although, an abstract service is specified by an I/O-automaton it cannot be implemented directly, but rather must be refined by other services – it only helps to structure services and, particularly, dependencies between them.

The formal definition of an abstract service is based on the refinement relation between two services. Intuitively, an abstract service has fewer legal inputs and/or more defined outputs than the refining service. This means that for a certain input stream the set of corresponding output streams of the refining service R is a subset of the output streams of the abstract service A. Additionally, $Dom(A) \subseteq Dom(R)$ is required. In particular, R gives a more concrete/restricted specification of the function modeled by A since it acts more deterministically concerning its I/O relation. Our definition of refinement permits to change the number of ports and their types in a specific way. For the formal definition see [4].

For example, the service *Manual Adjustment* from Figure 3 refines the abstract service *Adjustment*. The Service *Adjustment* has only one input and one output port, and specifies an abstract seat adjustment functionality: It receives a message defining the movement direction ($d1$ or $d2$) and sends a corresponding motor control message m that encodes the direction. *Adjustment* only specifies an immediate causality between its input and output messages: the user gives a direction to the DCU – the seat moves in the given direction. *Manual Adjustment* gives a more restricted specification of the adjustment function – it separates between length and height movement. The refining service has two input and two output ports, for the length and height movement directions respectively. In the Service Diagram from Figure 3, service *Memory* is another refinement of *Adjustment*. It receives an input message and moves the seat to a saved position, according to the stored height and length. The abstract service *Adjustment* will not be implemented in the Service Network because its role is only to identify some common behavior between services *Manual Adjustment* and *Memory*. Both services have an important property in common – they are restricted due to the dependency relation between *Child Seat Detection* and *Adjustment* (see below).

Aggregation The aggregation relations allow to arrange individual services which have been specified independently into a service hierarchy. Thus, it greatly helps to reduce the complexity of the system functionality.

The *aggregation* is defined as a relation between a service and its sub-services. It directly reflects the idea that the functionality offered by a service can be subdivided into different sub-functionalities. Intuitively, a sub-service specifies a sub-functionality of its super-service, in contrast to a refining service which refines the functionality of the whole refined service. Formally, a service S is called a *sub-service* of a (super-) service C, if C refines S and $Dom(S) \subseteq Dom(C)$. With this definition we require that the super-service has to be defined for all inputs for which its sub-service yielded a defined output as well. Additionally, the output streams of the super-service may be more restricted compared to the corresponding ones of the sub-service because of influences by other sub-services.

A super-service composed of several sub-services is called a *compound service*. Thus, according to the aggregation relation, we define the semantics of a compound service as a container of all *concurrently* operating sub-services. We do not specify the compound services by I/O-automata,

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

because their behavior can be reproduced from their sub-services using the well-defined semantics of the aggregation relation. Thus, the automaton of *Manual Adjustment* does not exist in the Service Diagram because it can be reproduced from the automata of sub-services *Length* and *Height*. In this example, the sub-services of the compound service are completely independent of each other – each of them can run without any impact on the other service. In other cases, some of the sub-services of a compound service may influence each other. Then, additionally to the aggregation, their mutual dependencies must be defined, because a sub-service may be restricted by other sub-services when they are combined in one compound service. For example, since the service *Child Seat Detection* influences the services *Manual Adjustment* and *Memory*, we have to specify this dependency in addition to the aggregation relation between compound service *Seat* and sub-services *Heating*, *Manual Adjustment*, *Memory* and *Child Seat Detection*. See the following paragraph for further details.

Dependency Aggregation and abstract services represent hierarchical relations between services. Although the structure of services is essential for understanding the user functionality of a system, dependency relations among services also have significant implications in the development of a system. By *dependency relations*, we mean relations between services in a way that the operation of one service depends on those of other services. Although there are a lot of methodological significant dependency relations like *enables*, *modifies* or *needs*, the scope of this paper is to approach the specification technique of these relations rather than to completely enumerate them.

In our example (cp. Figure 3), the service *Child Seat Detection* modifies the behavior of the abstract service *Adjustment* and, hence, of *Manual Adjustment* and *Memory*. If *Child Seat Detection* detects a child seat mounted on the front seat, both adjustment services, *Manual Adjustment* and *Memory*, are prevented to move the front seat according to their modular specifications.

As already mentioned, relations between services are specified as being observable from the overall system boundaries without changing their interfaces (no additional ports are added) and without characterizing the communication between them (no additional channels are added). Also, the modular behavior specification of single services (in our case I/O-automata) are not modified in order to realize a dependency relation between them.

To specify these relations we introduce additional constraints. A *constraint* restricts the behavior of the influenced service by defining dependencies between its I/O message streams and those of the influencing service. These constraints are defined by predicate logic expressions over names of services, ports as well as access operations and specify dependencies between port values of different services in time intervals.

For example, the service *Child Seat Detection* (CSD) permanently receives a message (yes or no) from the environment through its port in whether a child seat is mounted or not. If the message is yes, service *Adjustment* (A) is not allowed to move the front seat, i.e. to send a message through its port out. The dependency between both services is specified by the following constraint:

$$\forall t \in \mathbb{N} : (CSD[in](t) = yes) \Rightarrow (A[out](t) = [\,])$$

This means, for every point in time, if message yes is received on the port in of service *Child Seat Detection*, no message is sent on the port out of service *Adjustment*.

As our example illustrates, abstract services implicitly impose their constraints on the refining services. All constraints on an abstract service have to hold for its refining services. For our example this means that the constraint on *Adjustment* has to hold for *Manual Adjustment* and *Memory*, too.

It should be noticed that the constraints do not modify the modular specification of services. They only specify the interplay between them which must be satisfied in the Service Network (see Section 5.3). In other words, constraints provide criteria to verify the models of the consecutive layer. Only those models which do not violate these constraints are the candidates for valid models.

With the Service Diagram we introduced an adequate model for the specification of the user-visible functional requirements of the system under consideration. Hereby, the basic ideas are to reduce the complexity of the overall functionality by describing each of its functionalities independently by simple services (by means of I/O-automata), arrange these services into a service hierarchy (by means of aggregation and abstraction relations) and specify relationships between these individual services that show how they influence each other (by means of dependency relations).

5.3 Service Network

So far, the focus lay on a consistent description of the behavior as it is observable at the outer boundaries of the system. However, now we address the question how the black-box behavior (specified in the Service Diagram) can be realized by a network (*Service Network*) of communicating entities.

As we are still not necessarily interested in complete descriptions of the system behavior, again services are used as basic building blocks. But in contrast to the Service Diagram, in a Service Network the services communicate with each other in order to realize the demanded behavior. Architectural concerns on the other hand like the mapping from

services to components and designing a hierarchical component architecture are not considered at this level but delayed to the following layer.

The structural relations which were dominating in the Service Diagram mainly served the structured gathering and understanding of the system functionality. Thus, they are of no relevance in the Service Network. More precisely, in a Service Network the services are not hierarchically arranged and there exist neither abstract nor compound services. As mentioned in Section 5.2, the behavior specified by an abstract service is completely realized by its refining services. Compound services were basically used in order to structure the functionality. Since their behavior can be reproduced from their sub-services (in consideration of the relevant dependencies), it is enough to implement the refining services or sub-services respectively.

In our example only the services *Child Seat Detection*, *Memory*, *Length*, *Height* and *Heating* remain in the Service Network (see Figure 4). The compound services *Manual Adjustment* or *Seat* as well as the abstract service *Adjustment* must not be considered in the Service Network.

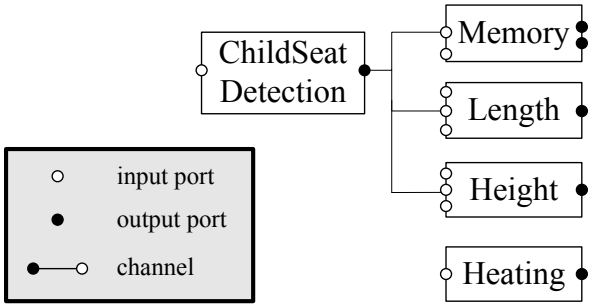


Figure 4. Service Network

However, it must be ensured that the resulting Service Network satisfies all dependencies in the Service Diagram. Therefore, dependency which are defined for abstract or compound services must be propagated to the refining services or sub-services and be reflected by their communication behavior.

The services of the Service Network refine the atomic services of the previous Service Diagram by adding communication behavior in order to realize the interplay between them. Thereby, each dependency relation of the Service Diagram results in a more or less complicated communication relation in the Service Network. This inter-service communication can be realized by adding internal channels in between the affected services. The syntactic interface and the behavioral semantics of the corresponding services must be extended accordingly. Sometimes, it can even be necessary or useful to introduce new services in order to realize the dependencies of the Service Diagram. In either case it must be assured that the behavior specified in the Service Diagram is completely realized in the Service Network.

In the following, both possibilities (adding new channels or new services) are explained in more detail. Note, that in both cases the syntactical interface of the overall system is not changed.

Internal Channels To establish communication between services, the interfaces of the services identified in the Service Diagram can be extended by additional internal ports. These can be used to link mutual depending services by internal channels. With *internal channels* we mean channels connecting two services, but not a service with the environment. Internal channels can only be attached to *internal ports*. In our example the functional dependency between *Child Seat Detection* and *Adjustment* (cp. Figure 3) is implemented by adding directed, internal channels connecting the service *Child Seat Detection* with all the services refining *Adjustment* (cp. Figure 4): the services *Length*, *Height*, and *Memory*. Via these channels the *Child Seat Detection* signalizes by a binary signal (*yes*/*no*) if a child seat is detected or not. Therefore, the interface of *Child Seat Detection* must be extended by an additional output port and the interfaces of *Length*, *Height* and *Memory* by respective input ports. Moreover, the behavior of these services must be adapted accordingly. The services *Length*, *Height* and *Memory* can execute their original behavior only if the messages received at the input ports connected to the service *Child Seat Detection* signalize that there is no child seat mounted.

Internal Services As mentioned before, it is sometimes useful to introduce new internal services. *Internal services* have only internal ports, i.e. ports which are not observable at the outer system boundaries. For example, priorities can be easily modeled in a Service Network that way. Let's assume that there would be the following additional dependency specified in the corresponding Service Diagram: The services *Length* and *Height* are mutual exclusive and *Length* has a higher priority than *Height*, i.e. if both length and height movement are demanded at the same time, the length movement will be executed. This relation can be implemented easily by adding an internal service *Multiplexer* which receives both input signals and decides if they are forwarded to the services *Length* and *Height* respectively.

Summarizing, the Service Network provides a specification of the overall system functionality (as specified in the services *and* constraints of the Service Diagram) only by means of communicating services. Thus, the Service Network constitutes a model that can be simulated easily. For example, all introduced concepts are directly supported by

the CASE tool AUTOFOCUS [21]. Besides, the Service Network can be used as starting base for the design of a Logical Architecture as introduced in the next section.

5.4 Logical Architecture

Changing the view from a pure black-box to a white-box view on the system the Service Network is the first step towards a logical system architecture. However, it is not dealt with architectural questions, and the specification of the system behavior is consistent but still incomplete. Thus, the most important task on this layer (*Logical Architecture*) is the totalization of the behavior. The behavior has to be defined completely and deterministically (i.e. the system provides a well-defined predictable output for each possible input sequence). As a consequence, the building blocks of the Logical Architecture are *components* with totally defined behavior in contrast to services. Therefore, the services are grouped together and mapped onto (hierarchical) components. Thereby, one or several services of the Service Network are related to one component of the Logical Architecture, i.e. a component can provide several services.

The second major design decisions we make on this layer is the design of an appropriate logical system architecture. The formation of the Logical Architecture can be influenced by different criteria, for example the communication relations of the Service Network, the hierarchical structure which has been developed in the Service Diagram, or certain non-functional requirements. Depending on which criteria are considered to be more important, the resulting Logical Architecture will turn out differently.

Though, the concrete definition of the Logical Architecture is essential for the development of a system, it is beyond the scope of this paper and subject of future work. Here, we just briefly sketched how the presented service models integrate in the overall software development process.

In this section, we presented three consecutive layers, namely the *Service Diagram*, the *Service Network* and the *Logical Architecture*, for the modeling of the functionality of a reactive multi-functional system. Note, that with the Logical Architecture the functionality of the system is completely described. Subsequent design models will not add any new functionality but only deal with the question how this functionality can be implemented adequately.

6 Related Work

The approach presented in this paper introduces an integrated model for both functionality specification and functionality hierarchical structuring as part of the layer framework. Thus, related work can be mainly found in three different areas: techniques for feature specification, formalization of feature models, and model based development.

Feature Specification A large number of contributions have been made over the past decade in order to specify multi-functional systems. Feature-oriented development methods, as for example Feature-Oriented Domain Analysis (FODA) [15] or Feature-Oriented Reuse Method (FORM) [16] identify, classify and structure features as well as interactions between them. FODA and FORM introduced a graphical tree-like notation that showed the hierarchical structure of features. Since the introduction of FODA by Kang et al. in 1990, many different kinds of graphical notations [11, 9, 12, 17] have been proposed to extend this original notation. However, these approaches only focus on the modeling of relationships between features, using uninterpreted features as the corresponding base concept. The second deficit of these methods is that the absence of a formal semantics of the graphical notations prevents an automatic analyze of them. In contrast, here the behavior of single features as well as the semantics of their relationships are specified.

Formalization of feature models The definition of a formal semantics for feature models is not new. In [2], Batory and O'Malley use grammars to specify feature models. Sun, Li et al. define in [23] a formal semantics for the feature modeling language using first-order logic. The formalization of feature models with propositional formulas goes back to the work by Mannion [19], in which logical expression can be developed for the model, using propositional connectives by modeling dependency between requirements. In [10], Czarnecki et al. argue that cardinality-based feature models can be interpreted as a special class of context-free grammars. Another approach to specifying multi-functional systems is introduced by van Lamsweerde et al. In [24, 25] they propose formal techniques and heuristics for detecting conflicts from specifications of goals (requirements) and their interactions specified in LTL. As mentioned in the latter paragraph, the main deficit of these approaches is disregard for the behavior of single features. "As a consequence, these approaches focus on the analysis of dependencies, however abstracting away from the causes for these dependencies" [22]. In [8], Czarnecki and Antkiewicz recognize that features in a feature model are only merely symbols. They propose an approach to mapping feature models to other models, such as behavior or data specifications, in order to give them semantics. However, this approach only focuses on assets like software components and architectures. Our work focuses on formalizing user requirements and their analyze in the early phases of the development process. The closest approach to our work is a theoretical framework introduced by Broy [3, 4]

where the notion of a service behavior is formally defined. This framework provides several techniques to specify and to combine features based on their behaviors. However, this quite theoretical approach does not cover several relevant methodological issues what our work focuses on (techniques for building of service models and for the specification of inter-service relations).

Model based development Another related area to our work is model-based development (e.g. [1]), which aims at modeling every important aspect of a software system. A compilable and deployable model is an abstract representation of a system which interacts with its environment. An important work in this area is the generative software development [7] introduced by Czarnecki. This system-family approach focuses on automating the creation of system-family members: a system can be automatically generated from a textual or graphical specification. However, this approach as well as approaches like [20, 22, 21] focus on the construction of a specific solution (e.g. software architecture) without supporting the formal requirement specification. In contrast, we concentrate on the formalization of functional requirements and close the formal gap between requirements and architecture design in the early phase of the model-based development process.

7 Conclusion and Future Work

The presented concepts can be roughly summarized as follows: We introduced two consecutive service models which allow to specify the functionality offered by a system in detail and shortly sketched how they can be integrated in a layered development process. The Service Diagram focuses on the modeling and structuring of the user-observable functionality while the consecutive model, the Service Network, concentrates on the communication relation between the set of all services. Going from a structured view of the functionality (represented by services) to a less abstract level where the focus is on the interaction between all services represents a natural way of engineering a system in a top-down fashion: At first we identify (observable) services and try to structure them into a hierarchical relation. After that we focus on how this hierarchy can internally be realized by inspecting the intercommunication between all relevant (possibly not directly observable) services. Lastly, the resulting Service Network serves as an architectural guideline for the subsequent construction of a Logical Architecture. In general, the introduced techniques are applicable in all domains where reactive systems are designed according to the desired functionality. In particular, they are not limited to the automotive domain.

Why did we concentrate to model the functionality of a system? In the domain of reactive, multi-functional systems, representing the functionality of a system means to describe precisely what a system should do, i.e. the functionality represents the essence of a system. Together with non-functional requirements we obtain a precise specification of the desired system already at a very abstract level. Note that this is essential since this level of abstraction of a system is the basis for changes due to the evolution or extension of the system with a maximum of re-use.

Both models, the Service Diagram and the Service Network, integrate into the requirements engineering process and bridge the gap between usually informally specified (functional) requirements and formal design models with a well-defined semantics. Our intension is not to replace the informal modeling techniques for the early requirements engineering process, but in our opinion informal models alone are not sufficient.

For a model-based development process, a seamless transition between the models of different abstraction layers is essential. The same notion of a service as the basic entity for describing functionality provides the basis for a smooth transition between both service models, even though the focus of the models is different.

The well-defined semantics of the introduced models allows to perform an automatic analysis of system properties. This allows to comfortably deal with problems such as feature interaction [6], resolving discrepancies between conflicting functions and verification of the system's behavior. The identification of all relevant domain-specific dependencies is part of our future work.

The concept of a service represents a suitable instrument to model single functionalities in a modular fashion. Together with a well-defined meaning for the composition of several services we are able to ascribe the specification of the overall system behavior to the specification of individual (sub-) functionalities. This helps to master the complexity of multi-functional systems.

So far, we have pretended to use the introduced techniques only to model single systems. But the principle of representing a (single) system as a set of related functions can easily be taken to model a family of related systems which share a set of functions. In turn, this requires to integrate concepts like variability and alternatives into our models and to define their semantics. For our service models, the notion of refinement and aggregation provides an adequate semantical basis to smoothly incorporate these concepts. By this, both models are suitable for the development of software product lines. We consider the extension of our concepts to be applicable in these areas as future work.

There are several issues which we did not point out in detail in this paper: E.g. we did not precisely specify the transition from the Service Diagram to the Service Network nor address the concrete form of dependencies together with a precise meaning. The definition of service-patterns for the

modeling of certain scenarios would enrich the degree of application in industry. Thus, such definitions and descriptions are a matter of future work as well.

References

[1] Model-driven architecture, 2004. http://www.omg.org/mda.

[2] D. Batory and S. O'Malley. The design and implementation of hierarchical software systems with reusable components. *ACM Trans. Softw. Eng. Methodol.*, 1(4):355–398, 1992.

[3] M. Broy. Service-oriented systems engineering: Modeling services and layered architectures. In *FORTE*, pages 48–61, 2003.

[4] M. Broy. A theory for requirements specification and architecture design of multi-functional software systems. To appear.

[5] M. Broy and K. Stoelen. *Specification and Development of Interactive Systems: Focus on Streams, Interfaces, and Refinement*. Springer, 2001. ISBN 0-387-95073-7.

[6] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec. Feature interaction: a critical review and considered forecast. *Comput. Networks*, 41(1):115–141, 2003.

[7] K. Czarnecki. Generative software development. In *SPLC*, page 321, 2004.

[8] K. Czarnecki and M. Antkiewicz. Mapping features to models: A template approach based on superimposed variants. In *GPCE*, pages 422–437, 2005.

[9] K. Czarnecki and U. W. Eisenecker. *Generative programming: methods, tools, and applications*. ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 2000.

[10] K. Czarnecki, S. Helsen, and U. W. Eisenecker. Formalizing cardinality-based feature models and their specialization. *Software Process: Improvement and Practice*, 10(1):7–29, 2005.

[11] M. Griss, J. Favaro, and M. d'Alessandro. Integrating feature modeling with the RSEB. In *Proceedings of theFifthInternational Conference on Software Reuse*, pages 76–85, Vancouver, BC, Canada, 1998.

[12] J. V. Gurp, J. Bosch, and M. Svahnberg. On the notion of variability in software product lines. *wicsa*, 00:45, 2001.

[13] F. Houdek and B. Paech. Das Türsteuergerät – eine Beispielspezifikation. http://www4.in.tum.de/lehre/vorlesungen/ase/ss05/iese-002_02.pdf (in German), accessed 31.10.2006.

[14] I. Jacobson. Use cases and aspects - working seamlessly together. *Journal for Object Technology*, 2(4):7–28, July/August 2003.

[15] K. Kang, S. Cohen, J. Hess, W. Novak, and A. Peterson. Feature-oriented domain analysis (foda) feasibility study. Technical report, SEI, CMU, Pittsburgh, PA, November 1990.

[16] K. C. Kang, S. Kim, J. Lee, K. Kim, E. Shin, and M. Huh. Form: A feature-oriented reuse method with domain-specific reference architectures. *Ann. Softw. Eng.*, 5:143–168, 1998.

[17] K. C. Kang, J. Lee, and P. Donohoe. Feature-oriented project line engineering. *IEEE Softw.*, 19(4):58–65, 2002.

[18] N. A. Lynch and M. R. Tuttle. An introduction to input/output automata. *CWI-Quarterly*, 2(3):219–246, 1989.

[19] M. Mannion. Using first-order logic for product line model validation. In *SPLC*, pages 176–187, 2002.

[20] D. E. Perry and A. L. Wolf. Foundations for the study of software architecture. *SIGSOFT Softw. Eng. Notes*, 17(4):40–52, 1992.

[21] B. Schätz. Mastering the complexity of reactive systems: the autofocus approach. In *Formal methods for embedded distributed systems: how to master the complexity*, pages 215–258, Norwell, MA, USA, 2004. Kluwer Academic Publishers.

[22] B. Schätz. Combining product lines and model-based development. In *Proceedings of Formal Aspects of Component Systems (FACS 2006)*, 2006.

[23] J. Sun, H. Zhang, and H. Wang. Formal semantics and verification for feature modeling. In *ICECCS '05: Proceedings of the 10th IEEE International Conference on Engineering of Complex Computer Systems (ICECCS'05)*, pages 303–312, Washington, DC, USA, 2005. IEEE Computer Society.

[24] A. van Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *RE '01: Proceedings of the 5th IEEE International Symposium on Requirements Engineering*, page 249. IEEE Computer Society, 2001.

[25] A. van Lamsweerde, E. Letier, and R. Darimont. Managing conflicts in goal-driven requirements engineering. *IEEE Trans. Softw. Eng.*, 24(11):908–926, 1998.

A Theory for Model-based Transformation Applied to Computer-supported Preservation in Digital Archives

Thomas Triebsees, Uwe M. Borghoff
University of the German Federal Armed Forces Munich
Department of Computer Science
85577 Neubiberg, Germany
{ Thomas.Triebsees | Uwe.Borghoff}@unibw.de

Abstract

Model transformation has applications in many areas as, e.g., in model-driven software development or automated knowledge exchange. When applied, model transformations usually have in common that the transformation process shall preserve *certain properties. In model-driven software development, e.g., abstract models are transformed into more specific ones while preserving the behavior of the specified software system. In this paper, we present a model transformation approach that implements a preservation-centric view. We introduce a formal preservation language that allows transformation algorithms to be computed automatically. Additionally, these generated transformations can be proven to respect specified preservation requirements. We demonstrate our approach using a systems specification example that stems from the digital archiving world. We specify a web-archiving system that consists of two components each of which uses its own website model for storage. Internal processes necessitate exchanging websites between these two components and, thus, model transformation. We apply our preservation language and show how the algorithm generation procedure generates a transformation algorithm that guarantees an unchanged external view on the transformed websites.*

1. Introduction and Motivation

Nowadays, computer-based systems are the key element in many application domains. As a central challenge, complexity and domain-specific characteristics of such systems have to be handled by modern software and hardware engineering methods. Our initial research in the field of digital archiving has shown that digital archives constitute a challenging application domain in this respect. Usually, digital archives manage large bodies of digital data containing diverse kinds of information and having different types of (semantic) relationships. For this purpose, archives mostly comprise a complex hardware and software environment. The crucial tasks of the underlying components is to ensure the preservation of digital material over a long (i.e., an essentially unspecified period of) time. The software and hardware architecture is therefore often highly modularized so as to facilitate component exchange when technology evolves. The interested reader is referred to the reference model for Open Archival Information Systems (OAIS) [2] to get an impression of the complexity of archiving systems.

Prevalent techniques to handle domain-related system characteristics comprise, among others, domain-specific modeling elements and languages (e.g., UML stereotypes and profiles, [5]). In order to handle system complexity and facilitate model re-use, the model-driven development approach (MDD) has been pushed forward since quite a while [4, 9]. There, computer-supported model transformation plays a key role, which is another commonality to the digital archiving world. In order to maintain availability and understandability of the stored digital objects, several archiving activities necessitate data re-processing and data exchange between components that may semantically affect object contents. On ingest, e.g., incoming data objects are usually adapted to an internal storage model, which may include addition of metadata and packaging. When making these objects available to the public or exchanging them between different archives, the dissemination process may include transformations to an exchange format. Since digital objects are stored in graph- or tree-based formats (e.g., XML), connections between model transformation in the MDD-world and format transformation in digital archiving are evident (cf. [3]). In digital archiving, however, a preservation-centric view is propagated. In the just-described example, archivists speak of internal *migration* processes where certain content or structure of digital objects shall be *preserved*.

Looking more closely at model-driven development and

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

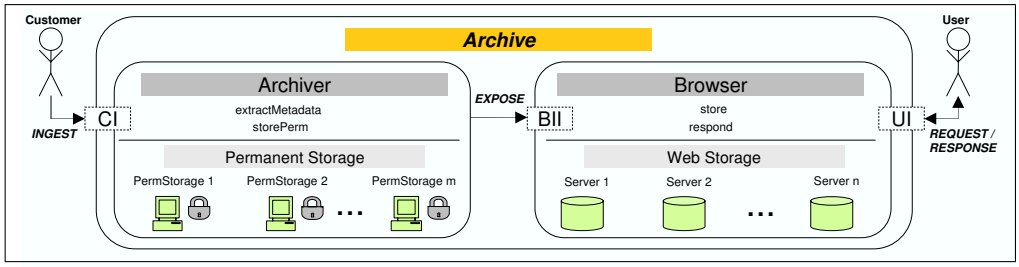


Figure 1. Example web archiving system

other applications fields of model transformation (e.g., ontology merging) one realizes that – while transforming the model – all these applications still aim at preserving given semantic invariants of the specified systems. These invariants, however, are usually implemented differently in the source and target model (we also say source and target *context*). In our previous work [11, 10] we have identified the usefulness of a declarative, high-level preservation language using a rich set of preservation constraints. Apart from briefly sketching how this language can be integrated into system specifications, this paper focuses on a comprehensive introduction of this approach as a theory for computer-supported model transformation that can be of general benefit to both - the digital archiving community and the systems engineering community. We will provide parts of the formal semantics of the preservation language and illustrate our approach. As case study we employ a real world example that stems from digital archiving. We informally introduce excerpts of the specification of a web-archiving system. The web archive consists of two components both serving complementary purposes. Therefore, these two components use different website models for storage which necessitates model transformation for data exchange. We use our language to formally specify the transformation task and preservation requirements. We show how the preservation constraints guide model transformation and even facilitate to generate transformation algorithms. As a consequence and major benefit of our formal approach, generated algorithms can be proven to work correctly w.r.t. specified preservation requirements.

The rest of the paper is organized as follows. In Sect. 2 we introduce the running example and informally describe the transformation task and preservation requirements. Sect. 3 introduces some terminology and the basic information model. In Sect. 4 we introduce the notion of a *concept*, which is used to group similar semantic relationships in different contexts. Sect. 5 is dedicated to formal preservation constraints that restrict permissible transformation results. We illustrate our approach (Sect. 6) by defining constraints for the running example and deriving a transformation algorithm that provably respects these constraints. Throughout the paper, we will visualize all theoretical parts by practical examples. We finally describe related work in Sect. 7 and close with a summary and a brief outlook in Sect. 8.

2. The Scenario - a Web Archiving System

Fig. 1 shows our example web archive. It consists of two components. The *Archiver* includes a permanent storage for long-term preservation of the hosted websites. The servers being annotated with locks indicate that access to this storage environment is not permitted from "outside" the archive. The latter is provided by the *Browser*-component, which includes a web-storage with fast server access. The archive has two external communication interfaces – the customer interface (*CI*) and the user interface (*UI*). Customers can ingest a website to the archive, which corresponds to the event *INGEST*. The *Browser* component provides a service for full website browsing as well as quick search facilities. Users can request these services by issuing a *REQUEST*-event. Upon request, they will receive the corresponding data as a *RESPONSE*-event.

2.1. Website model transformation between *Archiver* and *Browser*

The whole ingest-process affects both components. Having extracted metadata and stored the uploaded website as well as the extracted metadata permanently (*extractMetadata*, *storePerm*), the archiver sends the internal event *EXPOSE* to the browser's internal interface (BII). The browser then stores the website in the web-storage (*store*). The *Archiver* and *Browser* use different website models for storage. Hence, *EXPOSE* necessitates model transformation.

The *Browser*'s website model $BWeb$ is indicated on the left-hand side of Fig. 2. It essentially constrains folder content and folder structure. $BWeb$ comprises a source folder hosting exactly one file "index.html", the welcome page of the website. The names of the source folder and the target website shall match the content of the `<title>` element in "index.html". All html-files except "index.html" shall be located in the folder "html". We designate the folder $resources$ to contain all non-html-files.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

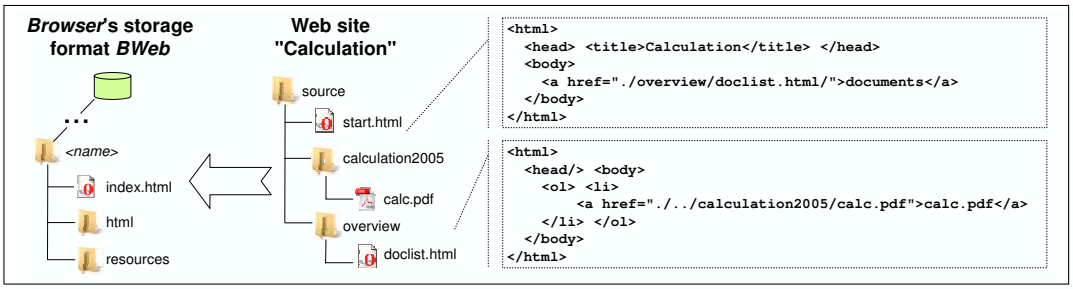


Figure 2. The source model instance "Calculation" and the target model $BWeb$

On the right-hand side, Fig. 2 depicts an example instance of the standard website model $Website$, which we want to transform to $BWeb$. The website "Calculation" consists of a folder structure containing two html-files and one pdf-file, where "start.html" is the welcome page. Fig. 2 also shows the contents of both html-files. While "start.html" contains the title of the page and a relative link to "doclist.html", "doclist.html" includes a list of linked resources containing "calc.pdf", only. This link is encoded relative to the folder structure, too.

When transforming "Calculation" from the standard website model $Website$ to $BWeb$, we want to preserve three major aspects. Most importantly, link consistency shall be preserved. This affects the relative links in "start.html" and "doclist.html". Moreover, we want to preserve the look and feel of the html-files, which corresponds to preserving most of the tag structure and contents of the hosted html-files. Clearly, the `href` attributes of the respective `<a>` tags will probably change during the transformation process. Finally, we wish to keep the name and content of "calc.pdf" unchanged.

In the sequel, these preservation requirements will be expressed using the preservation language. At this point we already see that its declarative nature facilitates the language to be easily integrated into, e.g., data flow specifications in systems specification theory. Notice that this example also exhibits some major challenges in model transformation. In particular, we have tree-based (i.e., recursive) structures (content of HTML-files) and transitive model properties (file containment in the folder hierarchy). Hence, from a theoretical point of view, this real-life example is compatible to the one used in [3] as reference example for model-driven development.

3. Basic Formal Information Model

Digital archives store digital objects that can be associated to each other. Object symbols $o \in O$ and constants $k \in K$ are uniquely typed within a type theory $(\mathcal{T}_O, \leq_O)$ with subtyping and a greatest type Obj. We distinguish three kinds of types — static types $\mathcal{T}_S \subseteq \mathcal{T}_O$, class types $\mathcal{T}_C \subseteq \mathcal{T}_O$, and collection types (e.g., sequences). Static and class types model static and dynamic parts of the archive, respectively. Constants, like the integer number 1 or the truth values T and F have a static type. In contrast, object symbols are place holders for data objects stored within the archive that can be created, transformed, and deleted. We regard object symbols $o \in O$ as IDs that are dynamically attached to objects on creation. Once used, an object's ID stays unchanged until the object is deleted. As a domain specific requirement, objects are immutable [6]. Thus, object transformations yield new objects with a new IDs.

Class types specify object templates by means of attributes. Fig. 3 depicts the most relevant excerpts of the source and target website model. Subtyping is indicated by the extension arrow known from UML. Type Tag, e.g., is recursive and comprises four attributes: a name $name : Tag \rightarrow String$, which is constrained to match the regular expression in braces, a sequence $attrs$ of attribute-value combinations, an attribute $content : Tag \rightarrow String$, and a sequence of subelements. A tag

$$t := < \texttt{a href} = \text{“./}\texttt{test.html}'' > \texttt{test} < /\texttt{a} >$$

is, thus, represented by

$$name(t) = \text{“}a\text{”}, attrs(t) = \langle(\text{“}href\text{”}, \text{“./}test.html\text{”})\rangle,$$
$$content(t) = \text{“}test\text{”}, \text{and } subelems(t) = \langle\rangle.$$

In addition to attributes, we support predefined archival functions $f \in F$. This may, e.g., include integer addition.

Objects may have labeled associations among each other with a fix signature and fix multiplicity constraints. Multiplicities are interpreted as usual. In Fig. 3, eP has signature $Website \times Folder \times HTML$ and multiplicity $1:1:1$. The associations $contFF$ and $contFFi$ model file structures.

All model elements are interpreted in algebras $\mathcal{A}$. Interpretations $o^{\mathcal{A}} \in O^{\mathcal{A}}, o \in O$ of object symbols and constants $k^{\mathcal{A}} \in K^{\mathcal{A}}, k \in K$ are called their *values* and must be an element of the *type domain* $\tau^{\mathcal{A}}$, in case $o : \tau / k : \tau$. If an object symbol has semantics $o^{\mathcal{A}} = \perp$, we say that o does not *exist* in the system. As in reality, the set of all existing objects has to be finite and associations may be set between existing objects, only. Function symbols and association symbols are interpreted as usual.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Notice the similarity of our information model to MOF-based models that are used for model transformation in model-driven development approaches.

3.1. Transformation algorithms

Transformation algorithms and subsequent states w.r.t. a transformation will play a key role when checking validity of transformation and preservation constraints in Sect. 5. Five basic operations can change system states: 1) object creation $\texttt{cre}(o)$, 2) object deletion $\texttt{del}(o)$, 3) setting an association $\texttt{set}(a,(o_1,...,o_n))$, 3) unsetting an association $\texttt{uset}(a,(o_1,...,o_n))$, and 5) object transformation $\texttt{tr}(\delta|o \mapsto o')$. There, $\texttt{cre}(o)$ and $\texttt{tr}(\delta|o \mapsto o')$ may be supplemented by a post condition. Moreover, $\texttt{tr}(\delta|o \mapsto o')$ and $\texttt{del}(o)$ can have a pre condition. Semantically, we consider the effect of all basic operations w.r.t. a preceding and a subsequent system state. As an example, we show the semantics for transformations:

$$\models (\mathcal{A} \rightsquigarrow \mathcal{A}', \texttt{tr}(\delta|o \mapsto o')), \text{ iff } o^{\mathcal{A}} \neq \bot \wedge o'^{\mathcal{A}'} \neq \bot \wedge \mathcal{A} \models pre(\texttt{tr}(\delta|o \mapsto o')) \wedge \mathcal{A}' \models post(\texttt{tr}(\delta|o \mapsto o'))$$
and no other changes to $\mathcal{A}$

$\mathcal{A}'$ is subsequent to $\mathcal{A}$ w.r.t. $\texttt{tr}(\delta|o \mapsto o')$, iff o exists in $\mathcal{A}$, o' exists in $\mathcal{A}'$ and o and o' satisfy the pre and post condition, respectively, of $\texttt{tr}(\delta|o \mapsto o')$. Notice that we prohibit all other changes to $\mathcal{A}$.

A sequence $\Delta := \langle op_1, ..., op_n \rangle$ of basic operations is considered a *transformation algorithm*. If Δ is empty, $\mathcal{A}$ and $\mathcal{A}'$ are subsequent w.r.t. Δ, if $\mathcal{A} = \mathcal{A}'$. Otherwise, we need a pair of states $\mathcal{A}_{i-1}, \mathcal{A}_i$ for each $op_i \in \Delta$ such that $\mathcal{A}_i$ is subsequent to $\mathcal{A}_{i-1}$ w.r.t. op_i.

4. Contexts and Concepts

Associations with multiplicities are insufficient to model semantic relationships. Association eP between objects $w : Website$, $f : Folder$, and $h : HTML$, for example, shall only hold if h is located in f. For objects $w' : BWeb$ we have even stronger constraints as mentioned in Sect. 2. Regarding this example it turns out that the entry point *concept* models the same intention (connect a website to its source folder and welcome page) for different *contexts*.

As an example, Fig. 4 shows the *concept template* $\mathcal{K}^{con}$ being the semantic supplement for $contFF$ and $contFFi$.

Analogous to programming languages, the *concept interface* KI^{con} defines arity and typing constraints for objects that may be in $\mathcal{K}^{con}$. In particular, Sig_{KI} is a sequence of variable-type-bindings and $\Phi_{KI^{con}}$ is a PL formula (Propositional Logic) restricting the type variables used in $Sig_{KI^{con}}$. $\Phi_{KI^{con}}$ assures that only two folders or a folder and a file are in $\mathcal{K}^{con}$. Formally spoken, a *variable assignment* $\eta := [o_1/x_1^o, o_2/x_2^o]$ is valid for KI^{con}, if $\Phi_{KI^{con}}[type(o_1)/\alpha_1, type(o_2)/\alpha_2]$ is a tautology. *Context templates* implement concept interfaces by means of an *embedding formula* ι in which the object variables x_i^o of the corresponding concept interface occur free. C_{dir}^{con} and C_{rec}^{con} define context templates for direct and recursive containment, respectively. Suppose we have a folder f and a file fi. Then $\iota_{C_{dir}^{con}}$ is only valid if 1) $x^a(f, fi)$ and 2) the full path of f concatenated with $fi.name$ equals the full path of fi. There, x^a is an association variable. Thus, a variable assignment η for C_{dir}^{con} has to include a substitution for x^a in order for $\iota_{C_{dir}^{con}}$ to be evaluable. In our example, only the substitutions $[f/x_1^o, f'/x_2^o, contFF/x^a]$ and $[f/x_1^o, fi/x_2^o, contFFi/x^a]$ suit for C_{dir}^{con} provided that $type(f) \leq Folder, type(f') \leq Folder$, and $type(fi) \leq File$. These two instantiations of C_{dir}^{con} show the benefit

Concept interface KI^{con}:	
$Sig_{KI^{con}}$	$\langle x_1^o : \alpha_1, x_2^o : \alpha_2 \rangle$
$\Phi_{KI^{con}}$	$\alpha_1 \leq Folder \wedge (\alpha_2 \leq File \vee \alpha_2 \leq Folder)$
Context template C_{dir}^{con}:	
$\Phi^O_{C_{dir}^{con}}$	$\alpha_1 \leq Folder \wedge (\alpha_2 \leq File \vee \alpha_2 \leq Folder)$
$\iota_{C_{dir}^{con}}$	$x^a(x_1^o, x_2^o) \wedge$ $fullPath(x_1^o) = fullPath(x_2^o) + +x_2^o.name$
Context template C_{rec}^{con}:	
$\Phi^O_{C_{rec}^{con}}$	$\alpha_1 \leq Folder \wedge (\alpha_2 \leq File \vee \alpha_2 \leq Folder)$
$\iota_{C_{rec}^{con}}$	$\exists f^o : Folder \bullet ($ $\mathcal{K}^{con}[\{C_{dir}^{con}\}](x_1^o, f^o) \wedge \mathcal{K}^{con}(f^o, x_2^o)$)
Concept template $\mathcal{K}^{con}$:	
$KI_{\mathcal{K}}^{con}$	KI^{con}
$C_{\mathcal{K}}^{con}$	$\{C_{dir}^{con}, C_{rec}^{con}\}$

Figure 4. Concept template for $\mathcal{K}^{con}$

of using type variables and association variables in context templates. That way we can "melt" overlapping contexts and avoid unnecessary duplicate work.

As shown by C_{rec}^{con}, we allow recursive concept definitions. A folder f or file fi is contained recursively within a folder f, if it is contained directly in f or there is a chain of folders starting from f such that neighbored elements are in $\mathcal{K}^{con}[\{C_{dir}^{con}\}]$ and the last element directly contains f/fi. The language construct $\mathcal{K}^{con}[\{C_{dir}^{con}\}]$ reduces evaluation of $\mathcal{K}^{con}$ to C_{dir}^{con}. To our experience, this feature can significantly speed up concept evaluation.

Concept satisfaction is defined as follows:

$$(\mathcal{A}, \eta, C) \models \mathcal{K}(o_1, ..., o_n), \text{ iff } C \in C_{\mathcal{K}} \text{ and } (\mathcal{A}, \eta[o_1/x_1^o, ..., o_n/x_n^o]) \models \iota_C$$

In Sect. 2 we have mentioned that websites have 1) a welcome page and 2) a source folder. These two items constitute a website's entry point. However, the semantic restrictions for objects of type $Website$ and $BWeb$, respectively, strongly differ. We do not provide the whole concept definition of the entry point concept $\mathcal{K}^{eP}$ for brevity, but rather describe the differences for the two contexts $C_{Website}^{eP}$ and

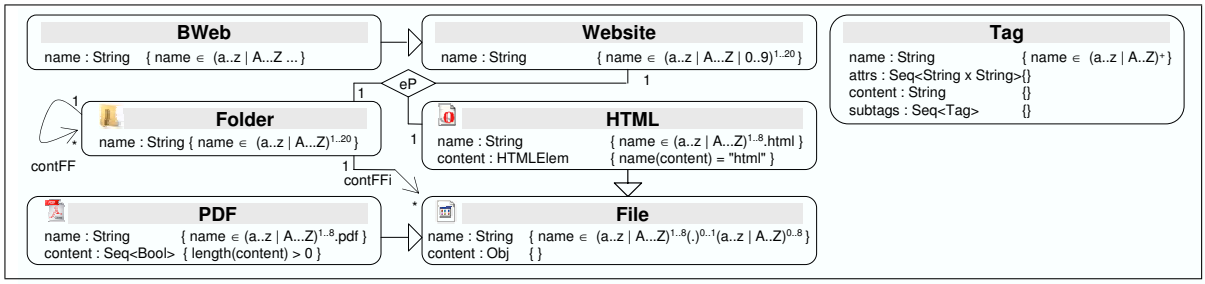


Figure 3. Formal source and target model $Website$ and $BWeb$

C^{eP}_{BWeb} in more detail than is done in Sect. 2. If a folder f and a file fi are the entry point for $w : Website$, $\iota_{C^{eP}_{Website}}$ constrains f to contain fi. If w, however, has type $BWeb$, then $\iota_{C^{eP}_{BWeb}}$ assures the following seven restrictions: 1) f directly contains fi. 2) fi has name "index.html". 3) f directly contains a "html"-folder, which may only contain html-files. 4) f directly contains a "resources"-folder containing non-html-files, only. 5) f contains the folders specified in 1.-4., only. 6) The welcome page is the only file directly contained by f. 7) The welcome page has a title tag t the content of which equals the names of f and of w.

When transforming the source model instance "Calculation" to the target model $BWeb$ in Sect. 6, we want to preserve $\mathcal{K}^{eP}$. Fig. 5 exemplifies this preservation task as a first indication of how we apply contexts and concepts in this respect. We, however, postpone a more detailed introduction to Sect. 5. In Fig. 5, C and C' denote $C^{eP}_{Website}$ and C^{eP}_{BWeb}, respectively. As shown, transformations are executed on the context level. The object triple (w, f, h) is transformed by $\delta_1 : Website \rightarrow BWeb$, $\delta_2 : Folder \rightarrow Folder$, and $\delta_3 : HTML \rightarrow HTML$. In case (w, f, h) satisfy $\mathcal{K}^{eP}$ in the source context C^{eP}_{w}, the corresponding relationship defined by $\iota_{C^{eP}_{wA}}$ shall hold for $\delta_1(w)$, $\delta_2(f)$, and $\delta_3(h)$ in the target context C^{eP}_{wA}. v and v' denote the truth values arising when evaluating ι_C and $\iota_{C'}$, respectively.

Fig. 2 shows that "start.html" points to "doclist.html" by means of a *relative* link. Again, linking is a wide-spread concept with different implementations in different contexts. Since links usually follow a given path structure, we use *Finite Automata* for link evaluation and construction. We will not go into detail with it for brevity. Parts of a link

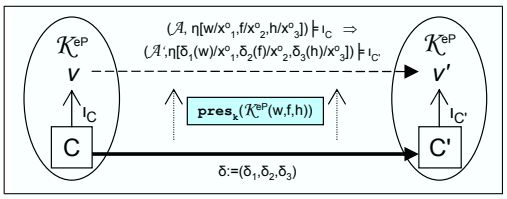


Figure 5. Preservation of $\mathcal{K}^{eP}$

automaton, which has been constructed automatically, are depicted in Fig. 9. A HTML-file h and a linked file f are in the linking concept $\mathcal{K}^L$, if h includes a `href`-attribute the value of which is a word that is accepted on a path through the corresponding link automaton A^L that ends in f.

5. Preservation

As explained in the introduction, our model transformation approach implements a preservation-centric view. Preservation, however, does not mean keeping contents unchanged. Transforming "Calculation" to $BWeb$, e.g., necessitates structural changes in the folder hierarchy. Hence, we surely have to adapt relative links. Preservation of this aspect means preserving link-consistency. More exactly, if a link pointing to a file fi is consistent before a transformation, it shall be consistent and point to the same (but possibly transformed) object after the transformation.

We use the tripartite approach briefly sketched in [10] to constrain desired transformation results: 1) transformation constraints, 2) object preservation constraints, and 3) concept preservation constraints.

5.1. Transformation constraints

First, we need a means to express a transformation intention. Only this facilitates to derive sensible transformation algorithms, i.e., such conforming to specified requirements. As a first account, we introduce *basic transformation constraints* of the form $o \mapsto \tau$, where o is an object symbol and $\tau \in \mathcal{T}_O$ is the target type. This constraint assures o is transformed to type τ. Concerning "Calculation", we demand $w \mapsto BWeb$, for example.

Validity of transformation constraints is defined via subsequent states as follows:

$$
\begin{array}{l}
(\mathcal{A}, \Delta, \mathcal{A}') \models o \mapsto \tau, \text{ iff } (\mathcal{A} \rightsquigarrow \mathcal{A}', \Delta) \wedge \\
\quad \exists o' : \tau' \bullet \exists n \geq 1 \bullet \\
\qquad \exists \mathtt{tr}(\delta_1 | o \mapsto o_1), \mathtt{tr}(\delta_2 | o_1 \mapsto o_2), \ldots, \\
\qquad\quad \mathtt{tr}(\delta_n | o_{n-1} \mapsto o_n) \in \Delta \bullet (\ o_n = o'\ )
\end{array}
$$

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

In order for $o \mapsto \tau$ to hold, we need an object $o' : \tau'$ existing in $\mathcal{A}'$ and a sequence $\delta_1, ..., \delta_n$ of transformations migrating o to o'. Existence of o, o' is guaranteed by the transformation sequence since transformations may map existing objects to existing objects, only (cf. Sect. 3.1).

Since it is impracticable to specify constraints for each single object, we support selection predicates. This yields *extended transformation constraints* of the form $\forall x^o : \tau \bullet (\ \phi \Rightarrow x^o \mapsto \tau')$, where only x^o may occur free in ϕ. These constraints are then transformed to basic constraints. When transforming "Calculation" in Sect. 6, we will use extended constraints.

5.2. Object preservation constraints

Object preservation constraints focus on object contents. The basic variant has the form $\texttt{pres}_o(o \mapsto \tau, o[\tau'])$ and is read as follows: "When transforming o to τ ($o \mapsto \tau$), preserve the content of o arising when o is abstracted to type τ' ($o[\tau']$)". The first parameter in parentheses and the parameter in brackets are optional. Hence, $o \mapsto \tau$ relates transformation constraints and object preservation constraints. If $[\tau']$ is omitted, τ' is implicitly set to $type(o)$. In this case, preserving o means preserving all attributes of o. Hence, the constraint $\texttt{pres}_o(p \mapsto PDF, p)$ is adequate to transform "calc.pdf" from our example website (cf. Sect. 2). Notice that $type(o) \leq \tau'$ and $\tau \leq \tau'$ must hold in case $[\tau']$ is specified. Otherwise, abstraction of o and $\delta(o)$ to τ' is undefined.

The semantics for basic object preservation constraints including all optional parameters is defined as follows:

$$
\begin{array}{l}
(\mathcal{A}, \Delta, \mathcal{A}') \models \texttt{pres}_o(o \mapsto \tau, o[\tau']), \text{iff} \\
\quad (\ (\mathcal{A}, \Delta, \mathcal{A}') \models o \mapsto \tau \ \Rightarrow \\
\quad\quad \forall x^o : \tau \bullet \forall n \geq 1 \bullet \\
\quad\quad\quad \forall \texttt{tr}(\delta_1 | o \mapsto o_1), ..., \texttt{tr}(\delta_n | o_{n-1} \mapsto o_n) \in \Delta \bullet \\
\quad\quad\quad\quad (o_n = x^o \Rightarrow o[\tau']^{\mathcal{A}} = x^o[\tau']^{\mathcal{A}'}\ )\ )
\end{array}
$$

A constraint $\texttt{pres}_o(o \mapsto \tau, o[\tau'])$ holds, iff the underlying transformation constraint holding implies o and $\delta(o)$ have similar contents when abstracted to τ'.

Obviously, transformation constraints specify *that* we want to transform an object, object preservation constraints specify what to preserve *if* an object is transformed. Therefore, having connected both via the first parameter in object preservation constraints is decisive.

As indicated above, we need an abstraction mechanism for object types and a comparison mechanism for object content. As proposed in [11], we use the relation $\approx_\tau$ to express *indistinguishability* w.r.t. a type τ. Constants are indistinguishable, if they are interpreted equally. Objects o and o' are indistinguishable w.r.t. τ if they are equal on all attributes of τ.

The following two type constructions are useful for abstraction. Suppose we transform $o : HTML$ to $o' : HTML$ and want to compare o and o' regardless of their $name$-attribute. Then $HTML - \{name\}$ producing a supertype of $HTML$ having only the $content$-attribute can be used to compare the tag-related contents of o and o'. If $o \approx_{HTML-\{name\}} o'$ holds, the `<html>`-tag of o and o' and all attributes thereof (including the subtags) are indistinguishable w.r.t. their types. As another construction we introduce type substitution. Considering type Tag in Fig. 3, $Tag - \{attrs\}$ constructs a supertype of Tag having no $attrs$-attribute. In case we want this construction be applied recursively, we append the substitution $[Tag - \{attrs\}/Tag]$. The whole construction $(Tag - \{attrs\})[Tag - \{attrs\}/Tag]$, which we abbreviate by $(Tag \ominus \{attrs\})$, henceforth, delivers the desired type.

5.3. Concept preservation constraints

Concept preservation constraints are the third pillar when specifying permissible transformation results. As they incorporate associations, they are a necessary supplement to transformation and object preservation constraints. Basic concept preservation constraints have the form

$$\texttt{pres}_k(\{o_{I_1} \mapsto \tau_1, ..., o_{I_m} \mapsto \tau_m\}, \mathcal{K}(o_1, ..., o_n), (C_s, C_t)),$$

where the first and last element in the outer parentheses are optional and $0 \neq m = \#I \subseteq \{1, ..., n\}$ is an index set. We read this constraint as follows: "When transforming o_{I_1} to τ_1, ..., and o_{I_m} to τ_m, then the transformation result shall 1) match a target context $C \in C_t$ and 2) satisfy $\mathcal{K}$ in C, if $\mathcal{K}(o_1, ..., o_n)$ was valid for a source context $C \in C_S$". In other words, preserving $\mathcal{K}$ means preserving the semantic connections defined by $\mathcal{K}$, but possibly in different contexts. Using C_s and C_t we can select desired source and target contexts, which can considerably speed up evaluation of $\mathcal{K}$. The index set I selects those objects that are connected to transformation constraints. Hence, only $o_1, ..., o_n$ may, but not all of them must occur in the set of transformation constraints. As already mentioned, we want to preserve $\mathcal{K}^{eP}$ when migrating "Calculation". $\mathcal{K}^{eP}$, however is defined differently for websites of type $Website$ and those of the target model $BWeb$. The constraint

$$\texttt{pres}_k(\{w \mapsto BWeb\}, \mathcal{K}^{eP}(w, f, fi), (\{C_w^{eP}\}, \{C_{wA}^{eP}\}))$$

assures the transformation result of w to have an entry point suitable for type $BWeb$.

The formal semantics

$$
\begin{array}{l}
(\mathcal{A}, \Delta, \mathcal{A}') \models \\
\texttt{pres}_k(\{\phi_1, ..., \phi_m\}, \mathcal{K}(o_1, ..., o_n), (C_s, C_t)), \text{ iff} \\
\quad \exists \eta \exists C \in C_s \bullet (\mathcal{A}, \eta, C) \models \mathcal{K}(o_1, ..., o_n) \ \Leftrightarrow \\
\quad \exists \eta' \exists C' \in C_t \bullet (\mathcal{A}', \eta' C') \models \mathcal{K}(o'_1, ..., o'_n), \text{ where} \\
o'_i := \begin{cases} \delta_j(o_i), & \text{if } \phi_j = o_i \mapsto \tau_j \wedge (\mathcal{A}, \Delta, \mathcal{A}') \models \phi_j \\ o_i, & \text{otherwise} \end{cases}
\end{array}
$$

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

defines satisfaction of concept preservation constraints w.r.t. a previous and a subsequent system state. We evaluate satisfaction of the concept $\mathcal{K}$ for the source and target objects using the set of allowed source and target contexts, respectively, and the set $\{\phi_1, ..., \phi_m\}$ of transformation constraints. Provided that there is a valid transformation constraint ϕ_j for an object o_i, the transformation result $\delta_j(o_i)$ is added to the target context. Otherwise, o_i is used. The strong correspondence $\Leftrightarrow$ postulated for concept satisfaction before and after transformation, respectively, might seem unusual at first sight. That way we, however, facilitate to specify that a transformation result shall not satisfy $\mathcal{K}$ if the source objects did not. Suppose we have a folder f and a file fi such that $\neg\mathcal{K}^{con}(f, fi)$ in the source context C. In order to preserve that status when transforming f to τ and fi to τ', we can specify $\texttt{pres}_k(\{f \mapsto \tau, fi \mapsto \tau'\}, \mathcal{K}^{con}, (\{C\}, \{C'\}))$.

6. Transforming the example $Website$ instance

In this section, we transform "Calculation" to the *Browser*'s storage format $BWeb$. Fig. 7 lists the corresponding transformation and preservation constraints, which seem quite bulky. In order to avoid defining them every time a website shall be transformed, one can specify profiles (as, e.g., proposed in [8]) generating the constraint set automatically whenever a $Website$ instance shall be transformed to $BWeb$. As another advantage, the constraint set scales well even for large websites, which underlines applicability of our approach.

The rest of this section is dedicated to algorithm generation. We have developed a formal procedure that generates algorithms working provably correct w.r.t. a given specification. We, however do not provide the formalism but rather exemplify it using our example. We successively construct a sequence of basic operations provably assuring validity of all constraints in Fig. 7. At this, we follow three guidelines: 1) Preserve nothing by default. 2) Construct as many objects as necessary, but as few objects as possible. 3) Process transformation and object preservation constraints first, concept preservation constraints last. When generating the transformation algorithm, we do not know what to preserve à priori. Moreover, certain parts will change during the transformation process and, thus, cannot or even must not be preserved. Therefore, we preserve nothing by default but rather demand the user to specify exactly those aspects he wants to preserve. The second strategy meets a central challenge in constraint solving — explosion of search space. Reducing the number of objects in the target website where possible, we reduce the search space. When applied to large (collections of) source objects, this procedure can significantly speed up algorithm generation. Strategy 3) is somewhat heuristic. When generating a transformation algorithm, we are bound to the transformations specified by transformation constraints. Having a constraint $o \mapsto \tau$, for example, the generated algorithm has to include a transformation $\texttt{tr}(\delta|o \mapsto o'), o' : \tau$. The pre and post conditions of it, however, have to be generated according to the object and concept preservation constraints given. Since object preservation constraints — to our experience — impose more constraints on an object's value than concept preservation constraints, the former are prioritized.

Transformation constraints:

1. $w \mapsto BWeb$
2. $\forall h : HTML \bullet ($
   $\exists f : Folder, h' : HTML \bullet ($
   $\mathcal{K}^{eP}(w, f, h') \wedge (h = h' \vee \mathcal{K}^{con}(f, h)))$
   $\Rightarrow h \mapsto HTML)$

Object preservation constraints:

3. $\forall p : PDF \bullet ($
   $\exists f : Folder, h' : HTML \bullet ($
   $\mathcal{K}^{eP}(w, f, h') \wedge \mathcal{K}^{con}(f, p)) \Rightarrow \texttt{pres}_o(p))$
4. $\forall h : HTML \bullet ($
   $\exists h' : HTML \bullet ($
   $\exists f : Folder, h' : HTML \bullet ($
   $\mathcal{K}^{eP}(w, f, h') \wedge (h = h' \vee \mathcal{K}^{con}(f, h)))$
   $) \Rightarrow \texttt{pres}_o(h[(HTML - \{name\})[Tag \ominus \{attrs\}/Tag]]))$
5. $\forall h : HTML \bullet ($
   $\exists h' : HTML \bullet ($
   $\exists f : Folder, h' : HTML \bullet ($
   $\mathcal{K}^{eP}(w, f, h') \wedge (\neg(h = h') \wedge \mathcal{K}^{con}(f, h)))$
   $) \Rightarrow \texttt{pres}_o(h[HTML[Tag \ominus \{attrs\}/Tag]]))$

Concept preservation constraints:

6. $\forall ws : Website, f : Folder, h : HTML \bullet (ws = w \Rightarrow$
   $\texttt{pres}_k(\{ws \mapsto BWeb, h \mapsto HTML\},$
   $\mathcal{K}^{eP}(ws, f, h), (C^{eP}_W, C^{eP}_{WA})))$
7. $\forall h : HTML, t : Tag, fi : File \bullet ((containsTag(h, t) \wedge$
   $\exists f : Folder, h' : HTML \bullet ($
   $\mathcal{K}^{eP}(w, f, h') \wedge \mathcal{K}^{con}(f, h))) \Rightarrow$
   $\texttt{pres}_k(\{h \mapsto HTML, t \mapsto Tag\}, \mathcal{K}^{L}(h, t, fi), (C_L, C_L)))$

Figure 7. Constraints for "Calculation"

Fig. 6 visualizes the first four steps when processing the constraints of Fig. 7. In step one, $w \mapsto BWeb$ is assured by transforming w using the operation $\texttt{tr}(\delta_1|w \mapsto w')$. Since there is a $1 : 1 : 1$ association eP between $BWeb$, $Folder$, and $HTML$, we have to create two additional objects $f' : Folder$ and $h' : HTML$, respectively, and set $eP(w', f', h')$. That way we achieve a transformation

$$\Delta := \langle \texttt{tr}(\delta_1|w \mapsto w'), \texttt{cre}(f'), \texttt{cre}(h'), \texttt{set}(eP, (w', f', h')) \rangle.$$

The post conditions of the transformation and creation operations, respectively, assure the type invariants for $BWeb$, $Folder$, and $HTML$. In particular, the $name$-attributes of these objects conform to the respective regular expression.

In the next step, we process $h1 \mapsto HTML$ and $h2 \mapsto HTML$ deriving from constraint 2. Again, the transformations only assure the type invariants for the target objects. According to our strategy "As few objects as possible", we now try to replace h' with either $h1'$ or $h2'$. Since this is

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

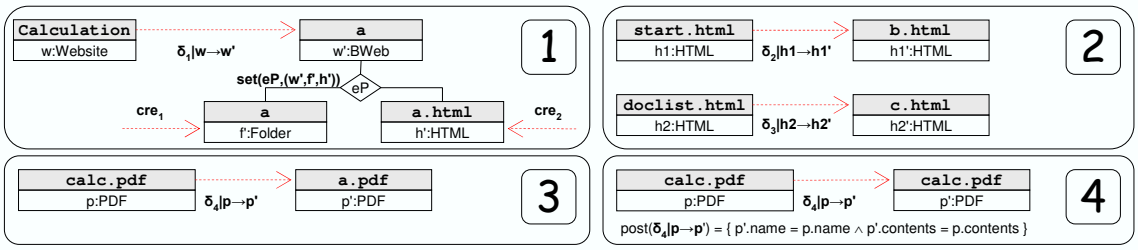


Figure 6. Algorithm generation steps 1 to 4

admissible, we generate two alternative transformations

$$\Delta_1 := \langle \mathtt{tr}(\delta_1|w \mapsto w'), \mathtt{cre}(f'), \mathtt{tr}(\delta_2|h1 \mapsto h1'), \mathtt{set}(eP,(w',f',h1')), \mathtt{tr}(\delta_3|h2 \mapsto h2')\rangle$$
$$\Delta_2 := \langle \mathtt{tr}(\delta_1|w \mapsto w'), \mathtt{cre}(f'), \mathtt{tr}(\delta_2|h1 \mapsto h1'), \mathtt{tr}(\delta_3|h2 \mapsto h2'), \mathtt{set}(eP,(w',f',h2'))\rangle$$

deriving from the substitutions $[\mathtt{tr}(\delta_2|h1 \mapsto h1')/\mathtt{cre}(h')]$ and $[\mathtt{tr}(\delta_3|h2 \mapsto h2')/\mathtt{cre}(h')]$, respectively. Notice that substituting $\mathtt{tr}(\delta_2|h1 \mapsto h1')$ by $\mathtt{cre}(h')$, is not permitted because the resulting transformation would not assure $h1 \mapsto HTML$. Moreover, $\mathtt{set}(eP,(w',f',h2'))$ is shifted behind δ_3 in Δ_2 since associations may be set between existing objects, only.

The third step introduces δ_4 for d assuring the respective type invariants. At this point, the two transformation algorithms respect all transformation constraints.

In step four, $post(\mathtt{tr}(\delta_4|p \mapsto p'))$ is extended such that $\mathtt{pres}_o(p)$ holds (constraint 3.). We achieve this by specifying equality of all attributes of type PDF for p and p'. Since we have modeled pdf-files as bit sequences, δ_4 constructs a target object with name and bitwise contents equal to p. The IDs, however, change.

Transforming the html-files is more complex. As an example, Fig. 8 shows how to assure

$$\mathtt{pres}_o(h1[(HTML - \{name\})[Tag \ominus \{attrs\}/Tag]]).$$

Type $HTML - \{name\}$ has exactly one attribute ($content$). Since Tag is no basic type and objects may be an attribute of at maximum one object, we have to transform $h1.content$. Clearly, this observation applies recursively and, thus, to all subtags of $h1.content$, as well. During the tag transformations, we have to preserve $Tag \ominus \{attrs\}$ so as to guarantee the overall preservation constraint. In Fig. 8, $\delta_5, ..., \delta_9$ transform the tags of "start.html". Notice that the `href`-attribute of the `<a>`-tag is not preserved. In order to demonstrate the effect of the type construction $(HTML - \{name\})[Tag - \{attrs\}/Tag]$ on the transformation output, we list the post conditions of $\mathtt{tr}(\delta_2|h1 \mapsto h1')$ and $\mathtt{tr}(\delta_6|t11 \mapsto t11')$ as an example:

$$post(\mathtt{tr}(\delta_2|h1 \mapsto h1')) = \{ h1'.content \approx_{Tag \ominus \{attrs\}} h1.content \}$$
$$post(\mathtt{tr}(\delta_6|t11 \mapsto t11') = \{ t11'.name = t11.name \wedge t11'.subtags \approx_{Seq(Tag \ominus \{attrs\})} t11.subtags \}$$

Instead of equality, we demand $h1'.content$ and $h1.content$ to be indistinguishable w.r.t. $Tag \ominus \{ attrs\}$. These post conditions are generated automatically from the underlying preservation constraint. Notice that the contents of the `<title>`-tag $t1111$ is preserved. Hence, $t1111'.contents =$ "Calculation". The second html-file is transformed analogously. In particular, the tag structure of "doclist.html" and the $content$-attribute of the `<a>`-tag is preserved. The latter assures the link to "calc.pdf" to appear equally when viewing "doclist.html" in a browser.

In the last steps, we turn to concept preservation constraints. In particular, we have to preserve the entry point and linking structure of w ($\mathcal{K}^{eP}$, $\mathcal{K}^{L}$). The upper part of Fig. 9 depicts the effects of constraint 6. For brevity, all following explanations will be given based on Δ_1 from above. In practice, the procedure introduced here would generate several possible algorithms respecting the constraints specified. First of all, we detect that $\mathcal{K}^{eP}(w, f, h1)$ holds in "Calculation", i.e., in the context C_w^{eP}. Moreover, w and $h1$ are transformed to w' and $h1'$, respectively, in Δ_1. In contrast, f stays unchanged. In order to fulfill the underlying preservation constraint, we have to assure $\mathcal{K}^{eP}(w', f, h1')$ is valid in context C_{wA}^{eP} (cf. Sect. 5.3). According to $\iota_{C_{wA}^{eP}}$ this implies that $f.name$ has to equal the value of the `<title>`-tag ("Calculation") contained by $h1'$. Since this does not hold, objects are immutable, and the content of the title tag shall be preserved (constraint 4.), we have to transform f. This transformation has not been specified initially, but is a mandatory *transformation consequence* as to preserve $\mathcal{K}^{con}$. In order to create as few objects as pos-

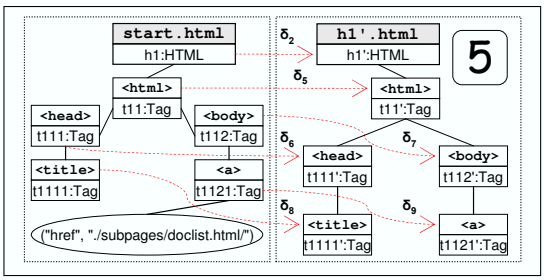


Figure 8. Algorithm generation step 5

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

sible, the algorithm generation procedure replaces $\texttt{cre}(f')$ in Δ_1 by $\texttt{tr}(\delta_{11}|f \mapsto f')$ with post condition $\{f'.name = t1111'.content\}$. Since this post condition accesses $t1111'$, which is the transformation result of δ_8, δ_{10} is located after δ_8 in Δ_1. Analogously, $\texttt{set}(eP, (w', f', h1'))$ is shifted. If the string "Calculation" did not match the regular expression required for folder names, an error would be raised. $\iota_{C^{eP}_{wA}}$ necessitates several other adaptations of Δ_1. In particular, $w.name$ is set to "Calculation", the folders $f1'$, $f2'$ are created and the specified folder structure is guaranteed according to $\iota_{C^{eP}_{wA}}$. As another requirement we have $h1'.name =$ "$index.html$". Therefore, $post(\texttt{tr}(\delta_2|h1 \mapsto h1')$ is extended by $h1'.name =$ "$index.html$".

Notice that $h2'$ and p' are still not subsumed within the folder structure of w'. This is assured indirectly by preserving $\mathcal{K}^L$ (link consistency), which constitutes step 7 (lower part of Fig. 9). In the following we will only sketch this preservation task. As the source website has consistent relative links in "start.html" and "doclist.html", they have to be preserved. For that purpose, the `href`-attributes of the `<a>`-tags in the created html-files $h1'$ and $h2'$ are adjusted according to the path structure in the target website. This is not forbidden by any preservation constraint. The link automaton for the target context is constructed according to the folder structure starting at the source folder of w'. As to preserve link consistency, the link from $h1'$ to $h2'$ induces that f' contains $h2'$. Analogously, f' has to contain p'. The algorithm generation procedure now tries to place $h2'$ and p' directly in f'. This is possible according to $\mathcal{K}^L$. However, recall $\mathcal{K}^{eP}$. There, we specified all html-files to be located in "html" and all other files in "resources". So as to satisfy $\mathcal{K}^{eP}$, we locate $h2'$ and p' in $f1'$ and $f2'$, respectively.

Now we consider the link from "start.html" to "doclist.html" as an example. The `href`-attribute of the `<a>`-tag in "start.html" contains the relative path to "doclist.html". Since "start.html" ($h1$) has been transformed to "index.html" ($h1'$), we have to preserve the link concept for $h1'$ and $h2'$ in the target context. There, $h1'$ is the link source, and $h2'$ is the link target. Up to now, we do not know the value of the `href`-attribute in the `<a>`-tag of $h1'$. We construct this value using the link automaton A'_L generated for the target website. The excerpts of A'_L relevant for the link between $h1'$ and $h2'$ are depicted in Fig. 9. The word that is constructed by a run from $h1'$ to $h2'$ in A'_L delivers the desired link. Setting the $attrs$-attribute of $t1121'$ accordingly finally assures link consistency.

Herewith, algorithm generation is completed. We have applied our prototypical JAVA-based implementation to the running example. It generated one transformation algorithm the execution of which indeed constructed a target website satisfying the constraints specified. We have also tested this example with considerably more complex (static) websites, e.g., such containing circular link structures and having high

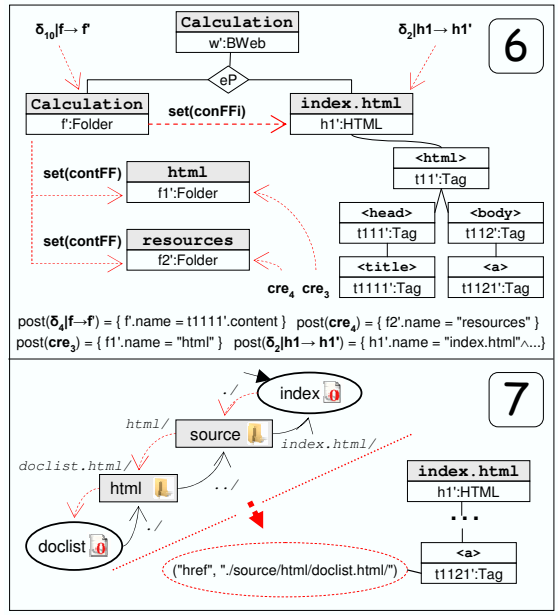


Figure 9. Algorithm generation steps 6, 7

degrees of connectivity. This resulted in up to 700 constraints that had to be satisfied. It turned out that a high degree of connectivity limits this approach since we have automated model construction on the target side. One possible bypass, however, is to group equivalent transformations (e.g., by their typing) and provide implementations for these equivalence classes. The preservation language then is used to model-check the transformation results w.r.t. the specification. Clearly, this significantly reduces the efficiency problem but shifts responsibility to the user. This section also indicated that constraint specification can be challenging. Therefore, we are currently working on a GUI for the framework's implementation. We are convinced that – if supported by appropriate graphical tools – users can get familiar with the preservation-centric way of thinking quite quickly. As an advantage, algorithms are generated iteratively. This facilitates to see the effects of specified constraints immediately. Hence, our approach can be used to let the user construct his preservation constraints interactively. Notice that user interaction may be necessary anyway since we use model construction based on full FOPL.

7. Related work

In the following we sketch the most relevant work, only. Our proposal is most closely related to model-based graph transformation [3]. We share the basic ideas to strictly constrain model instances via typed entities and association multiplicities and to use semantic matching points for transformation application. In contrast to graph transforma-

tion being transformation-centric, our approach, however, is preservation-centric. Using the preservation language, the user specifies desired semantic properties that shall be preserved. Through our formal notion of concept preservation we then can *derive* necessary transformations.

The research published in [1] concentrates on preservation *strategies*. Model elements are also interpreted in algebras. Objects themselves are modeled as state sequences representing the objects history. An object is preserved if it has the same interpretation in all states of its life cycle. We, however, have no strategic view but aim at generating transformation algorithms for one specific transformation step.

In [7] document *repositories* are examined. Dependencies among documents in different system states are defined using a language of first order temporal logic. A consistency checking strategy is developed that pinpoints inconsistencies and suggests repair actions. This work inspired us to use FOPL formulas for concept specifications.

The Typed Object Model (TOM, [12]) incorporates and deploys type descriptions via a type brokerage system. It focuses on file format transformations but works on a semi-formal level. We borrow the idea from TOM to base our notions for object preservation on subtyping.

8. Conclusion and Outlook

In this paper we have focused on a preservation-centric model transformation approach that has originally been developed to allow for computer-supported preservation in digital archives. We have argued that there are strong connections to other application areas as well, especially to the field of model-driven development. We pointed out that – for the application domains shown – transformations usually aim at preserving semantic invariants while transforming between different models. As a case study, we have used parts of the specification of a real-world web archiving system. We have applied our preservation language to constrain an internal model transformation process. Based on a formal information model, we have introduced transformation and preservation constraints that guide object transformations and even allow for an algorithm generation. As a benefit of our algorithmic view on transformation and the coherently formal approach, the generated algorithms can be proven to respect specified constraints.

Future work will cover three important issues. Up to now, constraints are evaluated iteratively. We believe, however, that a static analysis of the provided constraints will speed up our procedure. Here, a calculus for detecting implications among transformation and preservation constraints will be helpful. Moreover, we intend to make transformation less prone to unexpected interruptions by supporting more comprehensive definitions of consistent system states. Transformation algorithms then will produce consistent states as quickly as possible. Finally, we will examine hierarchical constraint specifications. This would be quite interesting for hierarchical systems specification and specification refinement. When, e.g., exchanging the *Browser* component of the running example, one surely wishes to preserve at least parts of the external view on the web-archive. We are convinced that our approach is applicable here and can be of benefit.

References

[1] J. Cheney, C. Lagoze, and P. Botticelli. Towards a theory of information preservation. Technical report, Cornell University, Ithaca, New York, 2001.

[2] Consultative Committee for Space Data Systems. Reference model for an open archival information system. Technical report, Space Data Systems, 2002.

[3] K. Ehrig, E. Guerra, J. de Lara, L. Lengyel, T. Levendovszky, U. Prange, G. Taentzer, D. Varró, and S. Varró-Gyapay. Model transformation by graph transformation: A comparative study. In *MTiP 2005, Int. Workshop on Mod. Trans. in Pract. (MoDELS 2005)*, 2005.

[4] P. Fraternali and P. Paolini. Model-driven development of web applications: the autoweb system. *ACM Trans. Inf. Syst.*, 18(4):323–382, 2000.

[5] S. Paunov, J. Hill, D. Schmidt, S. D. Baker, and J. M. Slaby. Domain-specific modeling languages for configuring and evaluating enterprise dre system quality of service. In *Proc. 13th IEEE Int. Symp. on Eng. of Comp. Based Sys. (ECBS'06)*, pages 196–208, Washington, DC, USA, 2006. IEEE Computer Society.

[6] PREMIS working group. Data dictionary for preservation metadata. Technical report, OCLC: Dublin, Ohio, RLG: Mountain View, California, 2005.

[7] J. Scheffczyk, U. M. Borghoff, P. Rödig, and L. Schmitz. Managing inconsistent repositories via prioritized repair actions. In *Proc. of the ACM Symp. on Doc. Eng. (DocEng 2004)*, pages 137–146, 2004.

[8] A. U. Schmidt and Z. Loebl. Legal security for transformations of signed documents: Fundamental concepts. In *Proceedings of the 2nd European PKI Workshop*, volume 3545, pages 255–270. Springer Verlag (LNCS), June 2005.

[9] T. Szemethy, G. Karsai, and D. Balasubramanian. Model transformations in the model-based development of real-time systems. In *Proc. 13th IEEE Int. Symp. on Eng. of Comp. Based Sys. (ECBS'06)*, pages 188–196, Washington, DC, USA, 2006. IEEE Computer Society.

[10] T. Triebsees and U. M. Borghoff. Preservation-centric and constraint-based migration of digital documents. In *Proc. of the ACM Symp. on Doc. Eng. (DocEng 2006)*, pages 59–62, October 2006.

[11] T. Triebsees, U. M. Borghoff, and J. Scheffczyk. Controlled migration in digital archives. In *Proc. of the Int. Conf. on Dig. Arch. Techn. (ICDAT2005)*, pages 5–19, June 2005.

[12] J. M. Wing and J. Ockerbloom. Respectful type converters. *IEEE Trans. on Software Eng.*, 26:579–593, July 2000.

DP-Miner: Design Pattern Discovery Using Matrix

Jing Dong, Dushyant S. Lad, Yajing Zhao
Department of Computer Science
University of Texas at Dallas
Richardson, TX 75083, USA
{jdong, dsl044000, yxz045100}@utdallas.edu

Abstract

Design patterns document expert design experience in software system development. They have been applied in many existing software systems. However, pattern information is generally lost in the source code. Discovering design patterns from source code may help understand system designs and further change the systems. In this paper, we present a novel approach to discovering design patterns by defining the structural characteristics of each design pattern in terms of weight and matrix. Our discovery process includes several analysis phases. Our approach is based on the XMI standard so that it is compatible with other techniques following such standard. We also develop a toolkit to support our approach. An industrial size case study is conducted to evaluate our approach and tool.

KEYWORDS
Design Pattern, Reverse Engineering, Matrix, XMI, UML, Design Pattern Discovery

1. Introduction

Large computer-based systems are normally difficult to understand and change due to lack of software architecture and design documentation. After the deployment of the systems, the original architecture and design information is generally lost. Source code becomes the only source to understand the systems. However, source code is typically large in size and hard to comprehend. It is very time-consuming and error-prone to read source code manually. Understanding the systems is very important since it may help on changing them. Software systems generally should be amenable to changes due to constant changes of user requirements, platforms, technologies and environments. Change is a constant theme of computer-based system design and development.

To understand the source code of a computer-based system, we need to recover the original architectural and design decisions and tradeoffs. Software design patterns [9] document expert design experience and may help capture design decisions and record design tradeoffs. Thus, discovering design patterns used in a software system may recover the original design decisions and tradeoffs and help on the understanding and future change of the system. Most of design patterns embed future changes that may only affect limited part of a design pattern. This evolution process can be achieved by adding or removing design elements in existing design patterns. When the pattern-related information is recovered from the source code, the developers are able to take advantage of the design patterns to change the design and thus the whole systems.

In this paper, we propose a novel approach based on matrix and weight to discover design patterns from source code. In particular, the system structure is represented in a matrix with the columns and rows to be all classes in the system. The value of each cell represents the relationships among the classes. The structure of each design pattern is also represented in another matrix. The discovery of design patterns from source code becomes matching between the two matrices. If the pattern matrix matches the system matrix, a candidate instance of the pattern is found. Besides matrix, we use weight to represent the attributes/operations of each class and its relationships with other classes. In addition to the structural aspect, our approach investigates the behavioral and semantic aspects of pattern discovery. A toolkit has also been developed to support our approach.

Different from other pattern mining approaches, our approach is based on the XML Metadata Interchange (XMI) [25] that is an interchange format for metadata in terms of the Meta Object Facility (MOF) [23]. XMI specifies how UML models [4] are mapped into a XML file. By representing a UML model in XML, the UML model can be searched for patterns. Thus, our pattern detection techniques can be naturally integrated with other techniques and tools following the XMI standard.

The remainder of this paper is organized as follows: the next section describes an overview of our approach. Section 3 presents the details of our approach in terms of several analysis phases. Section 4 discusses a case study on the Java.awt [22]. The last two sections cover related work and conclusions.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

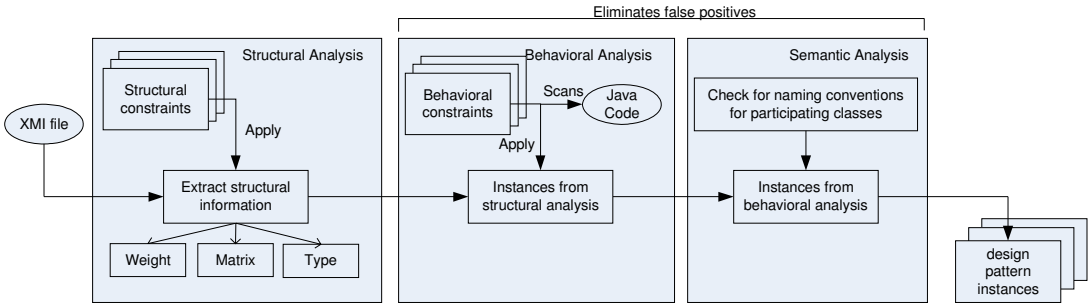


Figure 1 The Overall Architecture of the Approach

2. Overview of the approach

Figure 1 depicts the overall architecture of our approach to discovering design patterns in the source code of computer-based systems. Our approach uses the XMI standard as the intermediate representation format. The object-oriented systems can be initially reverse engineered into UML diagrams by existing UML tools, such as IBM Rational Rose [24]. Since UML diagrams are typically saved in proprietary formats of the corresponding UML tools, a standard XMI format of the UML diagrams has been defined and the plug-in of these UML tools has been developed to export UML diagrams into the XMI format. For example, the plug-in of IBM Rational Rose is called UniSys which can translate UML class diagrams into XMI format. Using the XMI plug-ins, UML class diagrams can be transformed into files in XMI format. These XMI files include all system design information, such as classes, attributes, operations, and different relationships between classes. Instead of analyzing the source code directly, our approach analyzes these XMI files.

Our approach consists of three phases: structural, behavioral, and semantic analyses. The structural analysis phase concentrates on the structural characteristics of the system, such as classes and their relationships. We introduce a novel approach that can extract structural information from software system design into weight, matrix and type. The structural aspect of each design pattern is also represented similarly by weight, matrix and type. Thus, the structural discovery of design pattern can be simplified by calculating and matching a group of numbers, instead of geometrical elements.

The results of the structural analysis may include the detected instances that are actually not a design pattern. Although such instances satisfy the structural characteristics of a design pattern, they may not be the instances of such design pattern due to missing behavioral characteristics. Such instances are false positives. Each design pattern generally may include both structural and behavioral characteristics. Thus, the behavioral analysis checks the results from the structural analysis for false positives. In this phase, our approach may check back the XML files or directly into the source code. Different from the structural analysis, such checks aim at verifying particular part of the system. Therefore, we do not need to go through the whole files.

It may be hard to distinguish some design pattern instances, such as the Strategy and Bridge patterns, since they have the same structural and behavioral characteristics. In such case, our approach analyzes the semantic information of the pattern instances. When a design pattern is applied into a system, the developers may follow some naming convention and include the role information of a pattern when naming the classes. Therefore, our approach checks whether the naming convention can distinguish design pattern instances.

We automated the structural, behavioral, and semantic analyses in our DP-Miner tool so that the users only need to generate the XMI input from existing tools.

3. Pattern discovery using matrix

Large computer-based systems normally include a large number of classes in the source code. These classes may relate to each other in a fairly complex manner. An instance of a design pattern is typically a group of classes that are structured and interacting with each other in a particular way. Detecting such structure and interactions in a large system can be challenging and time-consuming tasks. To efficiently discover design pattern instances in a large design, we encode the information about the classes and their relationships into weight and matrix. Our approach includes three phases: structural, behavioral, and semantic analyses.

3.1 Structural Analysis

In the structural analysis phase, we propose to use weight and matrix to represent the structural information of the system and the design patterns to be discovered.

In our approach, we take advantage of an important property of the prime number, i.e., the product of prime numbers is always a unique composite number. In other words, any given number can be broken into the multiple

of a unique list of prime numbers. Hence a single number can represent a unique combination of prime numbers. Based on this property we associate each structural element with a unique prime number. We can associate the most frequently used structural element with the lowest prime number other than 1. We exclude 1 because 1 is the identity for multiplication. We associate the prime numbers to the structural elements starting from 2 and come up with the following table:

Structural Elements	Prime number value
Attribute	2
Method	3
Association	5
Generalization	7
Dependency	11
Aggregation	13

Table 1 Prime Numbers of Structural Elements

Once we have every structural element associated with a unique prime number we can calculate the weight of each class and a matrix of the relationships between these classes. Both weight and matrix embed important information that allows us to integrate complex structural characteristics into numbers and matrices.

We define the weight of a class to be the multiples of the corresponding prime numbers to the powers of the total numbers of attributes, methods, and relationships associated with the class. The formula for calculating the weight of a class is:

$$W = w_a \times w_m \times w_{as} \times w_g \times w_d \times w_{ag}$$

$$w_a = 2^{(\text{number of attributes in the class})}$$

$$w_m = 3^{(\text{number of methods in the class})}$$

$$w_{as} = 5^{(\text{number of Association relationship of the class})}$$

$$w_g = 7^{(\text{number of generalization relationship of the class})}$$

$$w_d = 11^{(\text{number of dependency relationship of the class})}$$

$$w_{ag} = 13^{(\text{number of aggregation relationship of the class})}$$

where W represents the weight of a class. Once we compute the weight of the class we can easily get ideas about the structural elements of the class. If the weight of a class is 5670, for example, then this number is a unique combination of the prime numbers associated with the structural elements. Hence 5670 is broken into $2^1 \times 3^4 \times 5^1 \times 7^1$ which implies that there are one attribute and four methods in the class which has one association and one generalization relationships associated with it. In addition to relationships between classes, we consider the number of attributes and methods in each class because some design patterns require certain attributes and/or methods in their participating classes. For example, all participating classes of the Adapter pattern include at least one method as shown in Figure 6.

The value of W can become overflow due to potential huge value of the weight of the classes with a large number of attributes and methods. Optimizations are done to avoid such overflow conditions in Section 3.4.

Once we calculate the weight of a class which is symbolic of all structural elements, we can check whether a class in a system design belongs to a design pattern instance based on the single value of its weight. If a design pattern requires a particular class having minimum weight of 5670 then we can consider all classes with weight of 5670 or an integral multiple of it in a system design for that role of the design pattern. Thus, we can reduce the process of structural discovery of design pattern into simple arithmetical computations.

The relationships between classes are also critical in design pattern discovery. For each design pattern, there may be some relationships that have to exist. For this purpose we encode the relationships between the classes of a system design in a similar way as weight. In particular, we build a matrix that encodes the relationships between every two classes into a value of the cell corresponding to the two classes. In this way, the system design is encoded into an $n \times n$ matrix where n is the number of classes in the system. Similarly, the relationships in a candidate design pattern are also encoded into another $m \times m$ matrix where m is the number of participating classes in the design pattern. The discovery of design pattern is further reduced into the matching of the two matrices. To construct the matrix for both a system design and a design pattern we provide the following rules:

1. Initially the matrix is $n \times n$ where n is the number of classes involved. Each row and column represents a class arranged in the symmetric order, i.e., row i and column i must have the same class name. Each cell initially has value 1.
2. For each class i, if it has association relationship with another class j then we multiply the value of cell (i,j) by 5.
3. For each class i, if it has generalization relationship with another class j then we multiply the value of cell (i,j) by 7.
4. For each class i, if it has dependency relationship on another class j then we multiply the value of cell (i,j) by 11.
5. For each class i, if there is aggregation between siblings of class i to form an aggregate class j then the value at (i,j) is multiplied by 13.

To calculate the weight and matrix, we can find all necessary information from the XMI file generated from source code, except the dependency and aggregation relationships. We will deal with the dependency in the behavioral analysis phase. It is generally hard to distinguish the aggregation from association in reverse engineering processes since the difference between aggregation and association is generally at semantic

level, which is hard to distinguish at code level. In our approach, rule 5 is not applicable. Discovering aggregation depends on the interpretation of the intent of the developer. It is not a syntax based aspect hence we cannot detect the aggregation relationship based on the information provided in the XMI file from source code.

We apply the first three rules for constructing the matrix based on the XMI file. Once we created the weight and matrix from the XMI file of the source code we can check whether a particular class satisfies the requirements of a particular role of a design pattern by matching the weight and matrix of a design pattern with those of a system design. If the weights and matrix of a group of classes in a system design is integral multiples of those of the corresponding classes of a design pattern, this group of classes is considered as a candidate instance of the design pattern. Examples of weight and matrix calculations are presented in 4.2.

Besides weight and matrix, we check class type, i.e., if it is an interface, an abstract or a concrete class. Some design patterns may require their participating classes to be of certain types. If we can find a group of such classes, each of which satisfies a particular role of a design pattern, we record them as an instance of that design pattern. This list of pattern instances discovered from the structural analysis contains a number of false positives that are removed in the later analysis phases.

3.2 Behavioral Analysis

Each design pattern typically includes both structural and behavioral aspects. The structural aspect describes the static organization of the classes and their relationships in the pattern. The behavioral aspect presents the dynamic collaborations among the participants in the pattern. Thus, only structural analysis is not enough for discovering design pattern. The results from structural analysis may include false positive instances. Behavioral analysis can help eliminate such false positive cases. Based on the results from our structural analysis we further check the expected behavioral (dynamic) characteristics of each instance.

The main goals of behavioral analysis include finding the dependency that exists between classes and if some class delegates the method call. Similarly to the structural analysis, each design pattern may have different behavioral characteristics. This analysis can be fairly complex based upon these characteristics of the design pattern. One of the main difficulties in behavioral analysis is that there can be various possible implementations for similar expected behavior. One of such examples is the behavioral characteristic of checking the existence of a method call from one class to an overridden method in the subclass. It can be diagrammatically represented in Figure 2. This dynamic behavior can be implemented in the Component class in the following different ways:

1. ((FlipBufferStrategy)bufferStrategy).getCapablities();
2. bufferStrategy = new BltBufferStrategy(numBuffers, caps);
   bufferStrategy.getCapabilites();
3. BufferStrategy var=new BufferStrategy();
   var=(SingleBufferStrategy)bufferStrategy;
   var.getCapablities();

In all these three different implementations the behavioral characteristics are the same. There can be several other implementations as well. Hence the knowledge is hard to grasp at the implementation level. We need to abstract the idea behind the implementation which is the same for all different implementations.

To solve this problem of implementation variations, one of the solutions is to first represent the source code in a Control Flow Graph (CFG). This would later help in construction of sequence diagram for that code accurately, independent of the ways it is implemented. Recent work [16] has been done for static control-flow analysis and would be available as a plug-in for Eclipse in near future.

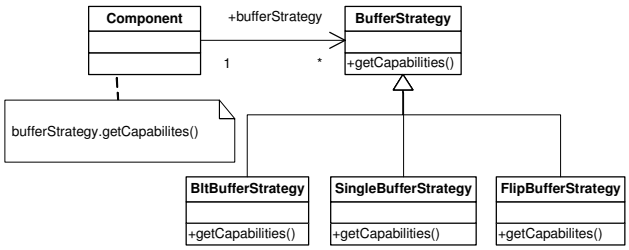


Figure 2 Instance of the Strategy pattern

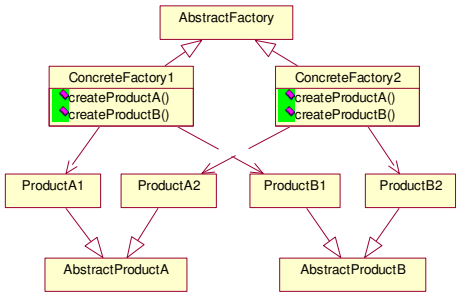


Figure 3 Abstract Factory Pattern Instance

In our approach, we try to identify these behavioral characteristics of each design pattern by scanning certain part of source code. Based on the results from our structural analysis, we have already found the candidates of design pattern instances. Thus, we can narrow down our search to certain files or classes, which can significantly reduce the scanning space.

For example, the dependency relationship is an important characteristic in some design patterns, such as the Abstract Factory pattern shown in Figure 3. This instance of the Abstract Factory pattern requires two groups of inheritance trees that are linked by

dependency between subclasses. To confirm such a pattern during behavioral analysis, hence, we only need to scan the ConcreteFactory1 and ConcreteFactory2 classes for such dependency.

3.3 Semantic Analysis

Most of the false positives can be eliminated in the behavioral analysis phase. However, there are certain closely related design patterns which are similar with respect to their structural and behavioral aspects but just different by their intent with which they were created. The Strategy and State patterns, Strategy and Bridge patterns are examples of such pairs of design patterns.

There are several ways for removing such ambiguous instances of design patterns from the results of the structural and behavioral analyses:

1. Searching the design documentation of source code.
2. Searching the in-line comments in the source code.
3. Getting clues from naming conventions of classes.

The first method generally works the best. However, the design documentation is not always available. It is quite common that the design documentation is lost, which is one of main reasons for reverse engineering. In the second method, in-line documentation generally describes the functionalities of a class, instead of pattern-related information. In addition, both the design and in-line documentation is likely to remain unchanged even when the corresponding code has been changed. The third method seems to be pretty relevant since the designers/developers are typically required to follow some convention when naming classes, attributes, and operations, whose names are generally not changed over time. Such naming convention sometimes may give us clues on their originally intent. For example, Figure 2 presents a pattern instance that we discovered from Java.awt based on our structural and behavioral analyses. Initially, our approach cannot distinguish whether it is an instance of the Bridge or Strategy pattern. After we investigated the case closely, we found the naming convention actually indicates it is an instance of the Strategy pattern since several class names include "Strategy" as substring. Thus, the Component class plays the role of Abstraction. The BufferStrategy class plays the role of Implementor. The BltBufferStrategy, FlipBufferStrategy and SingleBufferStrategy classes are the concrete implementors. Based on this study, we provide additional semantic analysis in our tool to check whether the naming convention has any clues to distinguish pattern instances that are similar in their structures and behaviors.

The limitation with this approach is that not all classes presented in a design pattern may be named with pattern-related information. In addition, the reverse engineering process is often applied to the implementation which is part of legacy code that might be developed without the pattern knowledge in consideration. Hence the naming conventions of classes might not always contain any clue for design pattern.

In our case study on the Java.awt package we found some classes include "Adapter" in their names. These classes are ContainerAdapter, ComponentAdapter, DragSourceAdapter, DropTargetAdapter, FocusAdapter, HierarchyBoundAdapter, KeyAdapter, MouseAdapter and WindowAdapter. Although these class names contain "Adapter", they are not qualified to be instances of the Adapter pattern due to two main reasons. First, the above Adapter classes are abstract classes instead of concrete classes. Second, there is no Adaptee presented in these instances. These Adapter classes are actually partial instances of the Adapter pattern, which are left open for future development or left for the users to provide their code for implementation of, e.g., the Adaptee class. As a result, we conduct the semantic analysis after structural and behavioral analyses.

3.4 Optimizations

In the previous sections, we present our novel approach on reducing pattern matching problem into weight and matrix calculations. In this section, we introduce some optimizations of our approach to deal with large cases. With a large number of attributes, methods and relations, the weight of a class can be a very large number that may cause overflow in practice. For software packages like Java.awt that contains 458 classes and 111 interfaces, the matrix construction time takes 28 seconds in our approach, which is not very good. For this purpose, we optimize our approach to improve scalability and performance in the following processes: (1) Weight calculation for class; (2) Matrix construction process.

Our original approach on calculating the cumulative weight of each class works fine for the applications with a small number of classes which have a small number of attributes, method and simple relationships among them. When we consider large software system, such as Java.awt, however, it has the scalability problem. The weight variable suffers an overflow. Consider the KeyEvent class with more than 150 attributes in Java.awt. Its weight is over 2^{150} which definitely results in overflow. To solve this problem, we optimize our weight calculation method.

Consider the design patterns to be identified; the number of attributes in any class participating in a design pattern is typically a small number. For example, the 23 patterns presented in [9] have at most one attributes in any class of each design pattern. Similarly, the number of operations in any class participating in a design pattern is also a small number, e.g., less than five

in all GoF patterns. If a class in an application has more than five operations, thus, it is not meaningful to calculate its weight in terms of all operations. Based on this observation, we optimize our weight calculation method by considering at most five attributes and operations in each class. If a class has more than five attributes/operations, we consider this class has five attributes/operations in our weight calculations. Hence the maximum weight of a class after calculating the attributes and methods is $2^5 \times 3^5 = 7776$.

For the relations between classes, we also optimize our methods based on the observation that two classes normally have only one instance of each relationship in a design pattern. For example, two classes may have at most one instance of generalization relationship. Although two classes in a software system may have multiple association and dependency relationships, there is generally only one instance of each kind of relationship between classes in a design pattern. Thus, it is not relevant to consider more than one instance of each kind of relationships in our weight calculations. Based on Table 1, the weight bases of the generalization, association, and dependency relationships are 5, 7, and 13, respectively. Hence, the maximum weight of a class becomes: $2^5 \times 3^5 \times 5 \times 7 \times 13 = 31352832$. This solves the problem of class weight overflow.

Similarly to the weight, we optimize the matrix calculation which represents the inter-class relationships. Since our matrix only encodes the relationships between classes, we consider only one instance of each kind of relationships when we calculate each cell of the matrix. Even though two classes of a software system may have more than one instance of the same kind of relationships, we only consider one instance of such relationship. For example, suppose ClassA and ClassB have three association relations starting from ClassA to ClassB and one generalization from ClassA to ClassB. Then the resulting value of the cell in the row of ClassA and the column of ClassB is $7 \times 5 = 35$ instead of $7^3 \times 5 = 1715$. This optimization is based on the similar facts as we restricted the number of attributes, methods and relationships during class weight calculation. Hence our approach restricts the maximum value of any cell of the matrix to $7 \times 5 \times 13$, i.e., 455.

Another optimization on calculating the matrix is to change the way we scan the XMI file. Initially, we scanned the XMI file by relationships, i.e., we look for all occurrences of one relationship between classes and multiply the corresponding cell before those of next relationship. Although it may save some time on calculating the matrix, it requires scanning the XMI file several times. In practice, we found that saving time on scanning the XMI file is more critical than calculating the matrix. Thus, we revised our algorithm to get the list of all possible relationships in which a class involves and then set the value of all cells in the row of the particular class depending on that list. In this way, we can calculate the matrix with a single iteration on the XMI file. These optimizations reduce the matrix construction time for Java.awt to 15 seconds.

4. Case Study

In this section we present a case study on the Java.awt package in JDK 1.4 using our approach to discovering design patterns. AWT stands for Abstract Windowing Toolkit, a powerful and flexible Java class library for the development of Graphical User Interface (GUI). It contains rich user components like Frames and Panels which ease GUI development and can run on any platform. The definition of each such component also contains a range of actions which can be performed on a specific event associated with the component. The AWT package also contains different layouts for placing components like Scrollbar, Buttons, etc., onto containers like Frame or Panel. These layouts do not depend on window size or screen resolution [22]. The Java.awt package consists of 346 files. These files contain a total of 485 classes (including inner classes) and 111 interfaces. It is large open source software.

In this case study on the Java.awt package we used a PC with 3.4 GHz Pentium processor, 1 GB of RAM and Windows XP operating system.

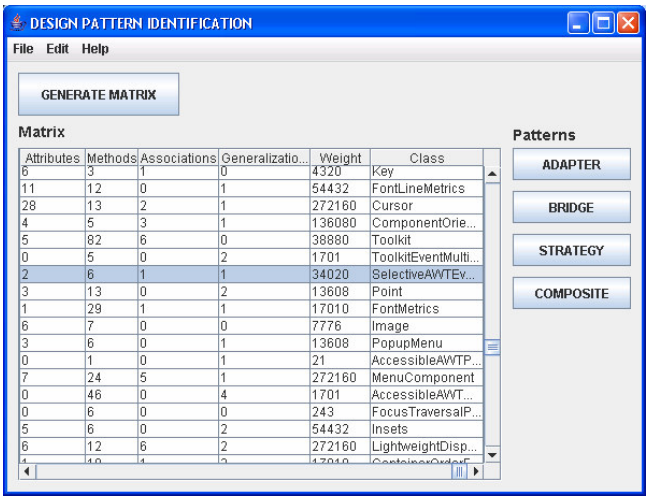
We developed a toolkit, DP-Miner, to support our approach described in the previous section. In this case study, we use DP-Miner to first calculate the weight of all classes and the matrix describing their relationships in the Java.awt package. We then describe the discovery of some design pattern instances based on our structural, behavioral, and semantic analyses.

4.1 Weight and Matrix

As discussed in the previous section, the weight of each class represents the number of attributes/operations in the class and the relationships with other classes. Figure 4 displays a screen shot of DP-Miner showing the weight of each class. The rightmost column lists all classes in the Java.awt package with their corresponding weights in the second rightmost column. These weights are computed based on the number of attributes, methods, outgoing associations, and outgoing generalization relationships, which are shown on the same row. For example, the weight of class SelectiveAWTEventListener is 34020, which contains 2 attributes, 5 methods, 1 association, and 1 generalization relationship. Only 5 methods are counted in the weight although Figure 4 shows it has 6.

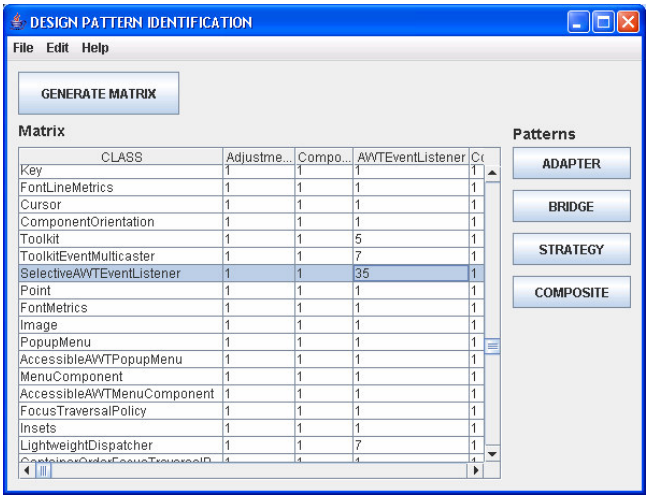
Some classes may have more than one outgoing generalization relationships because our current

approach treats the following two scenarios to be equivalent: (1) class i extends class j, (2) class i implements interface j. Therefore, a class may have multiple parent classes that are either an abstract class or interfaces.



Attributes	Methods	Associations	Generalizatio...	Weight	Class
6	3	1	0	4320	Key
11	12	0	1	54432	FontLineMetrics
28	13	2	1	272160	Cursor
4	5	3	1	136080	ComponentOrie...
5	82	6	0	38880	Toolkit
0	5	0	2	1701	ToolkitEventMulti...
2	6	1	1	34020	SelectiveAWTEv...
3	13	0	2	13608	Point
1	29	1	1	17010	FontMetrics
6	7	0	0	7776	Image
3	6	0	1	13608	PopupMenu
0	1	0	1	21	AccessibleAWTP...
7	24	5	1	272160	MenuComponent
0	46	0	4	1701	AccessibleAWT...
0	6	0	0	243	FocusTraversalP...
5	6	0	2	54432	Insets
6	12	6	2	272160	LightweightDisp...

Figure 4 Weight of all classes in Java.awt



CLASS	Adjustme...	Compo...	AWTEventListener	Cc
Key	1	1	1	1
FontLineMetrics	1	1	1	1
Cursor	1	1	1	1
ComponentOrientation	1	1	1	1
Toolkit	1	1	5	1
ToolkitEventMulticaster	1	1	7	1
SelectiveAWTEventListener	1	1	35	1
Point	1	1	1	1
FontMetrics	1	1	1	1
Image	1	1	1	1
PopupMenu	1	1	1	1
AccessibleAWTPopupMenu	1	1	1	1
MenuComponent	1	1	1	1
AccessibleAWTMenuComponent	1	1	1	1
FocusTraversalPolicy	1	1	1	1
Insets	1	1	1	1
LightweightDispatcher	1	1	7	1

Figure 5 Matrix of Java.awt

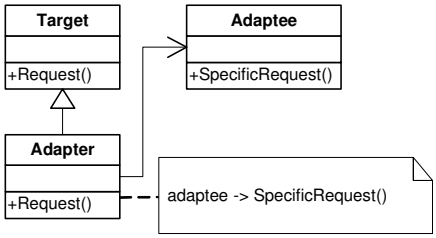


Figure 6 The Adapter pattern

Class	Target	Adapter	Adaptee
Target	1	1	1
Adapter	7	1	5
Adaptee	1	1	1

Table 2 The Adapter pattern matrix

Figure 5 presents the matrix that encodes the relationship information between all pairs of classes. All classes and interfaces are listed both at the top row and the leftmost column of the matrix. Each cell (i,j) represents the value of relationship from class i to class j. If there exists at least one association relation from class i to class j, the value of the cell (i,j) is set to 5. Similarly, if class j is a generalization of class i then the value of cell (i,j) is 7. Currently the aggregation and dependency relationships are not represented in the matrix thus the highest value of a cell can be $5^1 \times 7^1 = 35$.

Since the value of each cell in the matrix is the product of several prime numbers, we can decode the corresponding relationships from this single value of each cell. For instance, the value of the cell (SelectiveAWTEventListener, AWTEventListener) is 35 that equals to 5 × 7 in Figure 5. Thus, we know the SelectiveAWTEeventListener class inherits from AWTEventListener class. There is also an association relationship between them.

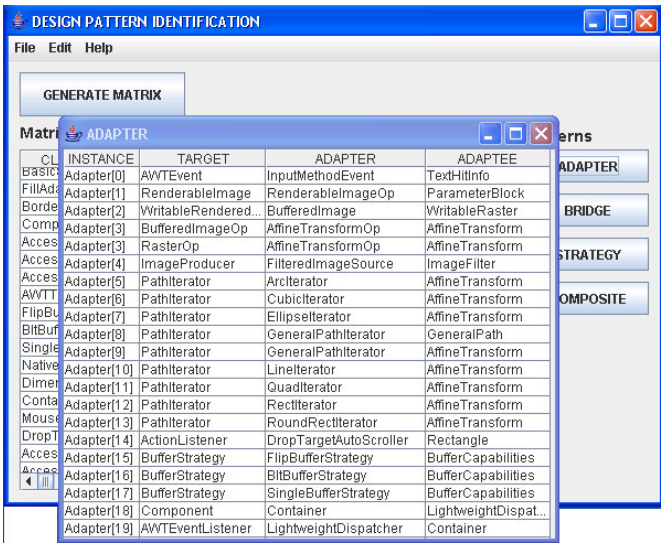
4.2 Pattern Discovery

Consider the discovery of the Adapter pattern whose intent is to convert the interface of a class into another interface clients expect. The class diagram of the Adapter pattern is shown in Figure 6.

In our structural analysis, we need to calculate the weight of each class in the Adapter pattern and the matrix describing the relationships between these classes. Based on our approach described in the previous section, the weight of Target is 3 (1 method), the weight of Adapter is 105 (1 method, 1 generalization and 1 association), and the weight of Adaptee is 3 (1 method). The matrix of the Adapter pattern is shown in Table 2.

Once we constructed the weights and matrices of both the Java.awt package and the Adapter pattern, we can find whether the package contains instances of the Adapter pattern following our approach described in the previous section. In particular, we first identify instances of Adapter based on the structural analysis, and then we can eliminate false positives based on the behavioral analysis. There are no ambiguity issues for Adapter hence the third phase is not applied.

The structural analysis becomes simply calculating whether any weights of a group of classes in Java.awt are integral multiples of those of the Adapter pattern as well as whether the corresponding cells of the Java.awt matrix are integral multiples of those in the Adapter matrix. The Adapter pattern matrix does not have to be exact sub-matrix of the Java.awt matrix, matching all corresponding cells.



INSTANCE	TARGET	ADAPTER	ADAPTEE
Adapter[0]	AWTEvent	InputMethodEvent	TextHitInfo
Adapter[1]	RenderableImage	RenderableImageOp	ParameterBlock
Adapter[2]	WritableRendered...	BufferedImage	WritableRaster
Adapter[3]	BufferedImageOp	AffineTransformOp	AffineTransform
Adapter[3]	RasterOp	AffineTransformOp	AffineTransform
Adapter[4]	ImageProducer	FilteredImageSource	ImageFilter
Adapter[5]	PathIterator	ArcIterator	AffineTransform
Adapter[6]	PathIterator	CubicIterator	AffineTransform
Adapter[7]	PathIterator	EllipseIterator	AffineTransform
Adapter[8]	PathIterator	GeneralPathIterator	GeneralPath
Adapter[9]	PathIterator	GeneralPathIterator	AffineTransform
Adapter[10]	PathIterator	LineIterator	AffineTransform
Adapter[11]	PathIterator	QuadIterator	AffineTransform
Adapter[12]	PathIterator	RectIterator	AffineTransform
Adapter[13]	PathIterator	RoundRectIterator	AffineTransform
Adapter[14]	ActionListener	DropTargetAutoScroller	Rectangle
Adapter[15]	BufferStrategy	FlipBufferStrategy	BufferCapabilities
Adapter[16]	BufferStrategy	BltBufferStrategy	BufferCapabilities
Adapter[17]	BufferStrategy	SingleBufferStrategy	BufferCapabilities
Adapter[18]	Component	Container	LightweightDispat...
Adapter[19]	AWTEventListener	LightweightDispatcher	Container

Figure 7 The Adapter pattern instances discovered

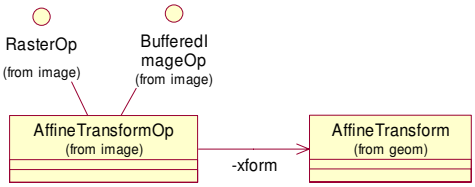


Figure 8 An Adapter pattern instance discovered

In addition to weight and matrix, we consider class type. In summary, we check the following structural characteristics in our structural analysis:

1. The Target class can be of type Abstract or Interface and its weight should be a multiple of 3.
2. The Adapter class should be of type Concrete and its weight should be multiple of 105. It should be the specialization of the Target class and have an outgoing Association relationship with Adaptee.
3. The Adaptee class should be a concrete class and its weight should be multiple of 3.

After the structural analysis, we conduct the behavioral analysis to remove false positive results in two steps. First our tool checks whether there exists any method that plays the role of the Request method, i.e., a method that is common in the Target and Adapter classes. The Request method is an important method that needs to be presented in the Adapter pattern. Without such method, the candidate instance cannot be considered as an instance of the Adapter pattern. If such method cannot be found in a candidate instance, that instance of Adapter is removed. Second our tool finds the occurrence of an invocation of a method in Adaptee from this Request method in the Adapter class. Even if a single such instance is found, the process of verification for that instance is terminated and moved on to the next instance. Otherwise, the candidate instance is removed since it is a false positive.

4.3 Results

Using DP-Miner, we discovered 21 Adapter pattern instances. The time taken for discovering these instances was 2.44 seconds after the matrix was generated. Figure 7 shows the snap shot of our tool displaying results of the Adapter pattern. Figure 8 shows the UML class diagram of the third instance of the Adapter pattern in Figure 7 found by our tool, where the RasterOp, AffinTransformOp and AffineTransform play the roles of the Target, Adapter and Adaptee, respectively. All operations in Interface RasterOp are implemented by AffineTransformOp. Hence they are candidates of the Request method of this instance. The "xform" variable is for the association relation between AffineTransform and AffineTransformOP. The following statement:

xform.transform(pts, 0, pts, 0, 4);

is defined in the method named "get2DBounds" in AffinTransformOp. Thus, the "get2DBounds" method plays the role of the Request method in the Adapter class, whereas method "transform" in AffineTransform plays the role of the SpecificRequest method in Adaptee.

We found Figure 8 actually contains two instances of the Adapter pattern with different Target classes but sharing the same Adapter and Adaptee classes. Hence another interface BufferedImageOp playing the role of Target is implemented by AffineTransformOp playing the role of Adapter. The following statement:

int type = xform.getType();

is defined in the "filter" method in AffineTransformOp. Thus, the "filter" method plays the role of Request, whereas the "getType" method plays the role of SpecificRequest.

One of the critical aspects of our discovery is the signature of the methods. The AffineTransformOp class implements two interfaces: BufferTransformOp and RasterOp. Both interfaces define the methods with the same name (called "filter") but different return types and parameters. The signature of the "filter" method defined in the BufferedImageOp interface is:

public final BufferedImage filter(BufferedImage src, BufferedImage dst)

The signature of the "filter" method defined in the RasterOp interface is:

public final WritableRaster filter(Raster src, WritableRaster dst)

Hence we need to take the return types and parameters of methods into consideration during the behavioral analysis. When confirming the second instance, DP-Miner checks if the "filter" method with BufferedImage, instead of WritableRaster, as return type invokes a method defined in AffineTransform. If the invocation does not exist in the proper "filter" method, the second instance is a false positive.

In a similar fashion, we applied our tool and found 3, 76, and 65 instances of the Composite, Strategy, and Bridge patterns, respectively, in the Java.awt package. We have manually checked the source code and document of Java.awt and found all detected patterns have correct pattern characteristics.

5. Related Work

Tsantalis *et al.* [18] applies an existing similarity score algorithm to detect design patterns. Each class relationship, such as generalization, creation, and delegation, is represented by a matrix. Every pattern characteristic is also presented by an $n \times n$ matrix. Thus, many matrices are used in their representations. The detection of design patterns is achieved by calculating the similarity matrices using the existing similarity score algorithm. In contrast to their multi-matrix approach, we encode all design information into a single matrix using prime numbers. Thus, our detection task can be reduced to simple arithmetic rather than complex matrix computations. Besides, our approach uses exact match instead of approximate match.

Balanyi and Ferenc [3] introduce a method to recover design patterns using a XML-based language, called Design Pattern Markup Language (DPML). The structural information of a design pattern, including the participating classes, their attributes and operations, and their relationships, is specified in DPML. The Abstract Semantic Graph (ASG) is constructed to represent the source code. Therefore, finding a design pattern in their approach is to find sub-structures in ASG which match its DPML descriptions. Their approach is different from ours in that they have two representations, ASG and XML for source code and pattern specifications, respectively. Pattern matching in their case is from one kind of representation to another. Our approach uses XMI as the intermediate representation for both source code and patterns without any issues with different representations. We also introduce weight and matrix.

Asencio *et al.* [2] develop a pattern recovery system called Osprey. C/C++ source code is parsed using Imagix. The information is stored in XML file since XML helps validating the output of their Imagix scripts. Their pattern recognizer, developed using a structural specification language based on set theory, can discover patterns from their internal representation of source code. Although XML is used, the main goal is to help validate the output of Imagix scripts, not as intermediate representation for pattern discovery since XML is converted into Osprey internal representation. In contrast, our approach directly uses XMI as the internal representation in our pattern recovery process.

Antoniol *et al.* [1] use the Abstract Object Language (AOL) as the intermediate representation for pattern discovery. To improve the recovery process, they extract class metrics, such as the numbers of attributes, operations, and relations, from AOL Abstract Syntax Tree (AST). In addition to the class metrics, we introduce weight and matrix that can be calculated from these metrics. In this way, we can integrate several metrics into a single value. Thus, our pattern discovery process is reduced into arithmetic computations.

Gueheneuc *et al.* [10] propose to fingerprint design pattern from source code using machine learning techniques. Size, filiations, cohesion, and coupling are their metrics. Classes that do not match these metrics are eliminated from the candidate set so as to reduce the search space. Our approach also uses metrics similar to size and filiations, but not cohesion degree and coupling degree. Instead of applying machine learning techniques, we reduce the searching problem into simple computation problem.

A top-down-bottom-up recovery approach based on FUJABA [21] platform and AST internal representation is presented in [14][15]. Both top-down and bottom-up search on AST work independently to check each other's search results to speed up the search speed. In an extension of FUJABA work, Wendehals [19] suggests using runtime data to check the behavioral aspect of patterns dynamically. The runtime information is gathered during program execution using a debugging tool. We use different intermediate representation and search process. Our search process consists different phases, each of which narrows down the search scope based on the input candidate set. Our behavior analysis does not need to manually collect runtime information. It is realized automatically.

Heuzeroth *et al.* [11] emphasize the importance of both structure and behavior of design pattern because neither structure nor behavior analysis alone is sufficient. AST is used as an intermediate representation of source code. Structure specification contains Prolog predicates, whereas dynamic constraints are defined as Prolog procedure based on temporal logic of actions (TLA) [12]. Our approach, on the other hand, uses different intermediate representation and considers not only structural and behavioral but also semantic aspects.

Keller *et al.* [13] use the SPOOL, an environment for reverse engineering of design components based on structural information extracted and represented using both UML metamodel and CDIF Intermediate Source Model. Their pattern identification is through query. The System for Pattern Query and Recognition (SPQR) is proposed by Smith and Stott in [17]. They do not hard-code pattern definitions, but infer pattern variants dynamically during code analysis with the help of a theorem prover. Both approaches use different techniques from ours.

6. Conclusions

In this paper, we presented a novel approach to reverse engineering design patterns from source code. We introduce weight and matrix that facilitate the pattern matching process as arithmetic computations. Different from other pattern mining approaches, our intermediate representation is based on XMI standard, which allows our techniques and tools to seamlessly integrate with other tools following such industry standard, such as IBM Rational Rose [24] and ArgoUML [20]. Our approach includes different analysis phases: structural, behavioral, and semantic analyses to reduce false positives. We also conduct a real-world case study to evaluate our approach and tool.

Our future work includes applying data-mining techniques on matching pattern weight and matrix to those of the systems and applying sparse matrix algorithms for storage and computation. Our approach currently can detect four patterns. Other patterns can be detected similarly, which will be included in our tool. In addition, we plan to integrate this work with our other work on design pattern compositions [5][6] and evolutions [7] to achieve round-trip pattern engineering. Furthermore, the detected design pattern instances can be visualized using our VisDP tool [8] to display the identified patterns in the context of the whole class diagram of the system. We plan to include our detection results as stereotypes in the input XMI file of VisDP because DP-Miner provides all necessary information.

References

[1] G. Antoniol, R. Fiutem, and L. Cristoforetti, "Design pattern recovery in object-oriented software." *Proceedings of the 6th IEEE International Workshop on Program Understanding (IWPC),* pp 153-160, 1998.

[2] A. Asencio, S. Cardman, D. Harris, and E. Laderman, "Relating expectations to automatically recovered design patterns." *Proceedings of the 9th Working Conference on Reverse Engineering (WCRE)*, 2002.

[3] Z. Balanyi and R. Ferenc, "Mining design patterns from C++ source code." *Proceedings of the 19th IEEE International Conference on Software Maintenance (ICSM)*, pp. 305-314, September, 2003.

[4] G. Booch, J. Rumbaugh, I. Jacobson. The Unified Modeling Language User Guide, Addison-Wesley, 1999.

[5] J. Dong, P. Alencar, and D. Cowan, Ensuring Structure and Behavior Correctness in Design Composition, *Proceedings of the 7th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems(ECBS),* pp279-287, Edinburgh UK, 2000.

[6] J. Dong, P. Alencar, and D. Cowan, Automating the Analysis of Design Component Contracts, *Software - Practice and Experience (SPE), Wiley,* Volume 36, Issue 1, pages 27-71, January 2006.

[7] J. Dong, S. Yang and K. Zhang, A Model Transformation Approach for Design Pattern Evolutions, *Proceedings of the Annual IEEE International Conference on Engineering of Computer Based Systems (ECBS)*, pp 80-89, Germany, March 2006.

[8] J. Dong, S. Yang and K. Zhang, VisDP: A Web Service for Visualizing Design Patterns on Demand, *Proceedings of the IEEE International Conference on Information Technology Coding and Computing (ITCC)*, pp385-391, USA, April 2005.

[9] E. Gamma, R. Helm, R. Johnson, J. Vlissides, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, 1995.

[10] Y. Gueheneuc, H. Sahraoui, and F. Zaidi, "Fingerprinting design patterns." *Proceedings of the 11th Working Conference on Reverse Engineering (WCRE)*, 2004.

[11] D. Heuzeroth, T. Holl, G. Hogstrom, and W. Lowe, "Automatic design pattern detection." *Proceedings of the 11th International Workshop on Program Comprehension (IWPC)*, pp 94-103, 2003.

[12] D. Heuzeroth, S. Mandel, and W. Lowe, "Generating design pattern detectors from pattern specifications." *Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE)*, 2003.

[13] R. K. Keller, R. Schauer, S. Robitaille, and P. Page, Pattern-based reverse-engineering of design components. *Proceedings of the 21st International Conference on Software Engineering (ICSE)*, pp 226-235. 1999.

[14] J. Niere, W. Schafer, J. P. Wadsack, L. Wendehals, and J. Welsh, "Towards pattern-based design recovery." In *Proceedings of the 24th International Conference on Software Engineering (ICSE)*, pp 338-348, 2002.

[15] J. Niere, J. P. Wadsack, L. Wendehals, "Handling large search space in pattern-based reverse engineering." *Proceedings of the 11th IEEE International Workshop on Program Comprehension (IWPC)*, pp. 274-279, 2003.

[16] A. Rountev, O. Volgin, M. Reddoch, *Static Control-Flow Analysis for Reverse Engineering of UML Sequence Diagrams,* 6th ACM Workshop on Program Analysis for Software Tools and Engineering, Lisbon, Portugal, 2005.

[17] J. M. Smith and D. Stotts, "SPQR: Flexible automated design pattern extraction from source code." In *Proceedings of the 18th IEEE International Conference on Automated Software Engineering (ASE)*, 2003.

[18] N. Tsantalis, A. Chatzigeorgiou, G. Stephanides, and S. Halkidis, "Design Pattern Detection Using Similarity Scoring." *IEEE transaction on software engineering*, Vol. 32, No. 11, November 2006.

[19] L. Wendehals, Improving design pattern instance recognition by dynamic analysis. *Proceedings of the ICSE workshop on Dynamic Analysis (WODA)*, 2003.

[20] ArgoUML, http://argouml.tigris.org/

[21] Fujaba User Documentation http://wwwcs.uni-paderborn.de/cs/fujaba/documents/user/manuals/FujabaDoc.pdf

[22] Java.awt resource information, September 2006, http://java.sun.com/j2se/1.5.0/docs/guide/awt/index.html.

[23] Model Driven Architecture. http://www.omg.org/mda/

[24] Rational Rose website. http://www.rational.com/

[25] W3C, Extensible Markup Language (XML), http://www.w3.org/

Introducing Impact Analysis for Architectural Decisions

Matthias Riebisch, Sven Wohlfarth
Department of Software Systems / Process Informatics
Technical University Ilmenau, 98684 Ilmenau, Germany
{ sven.wohlfarth | matthias.riebisch }@tu-ilmenau.de

Abstract

Architectural quality constitutes a critical factor for contemporary software systems, especially because of their size and the needs for frequent, quick changes. For success-critical business systems, architectural decisions are of high risk for the market share and even for the existence of enterprises. These decisions are important for design processes as well as for refactoring. Because of the complexity of the decisions, e.g., uncertain, contradicting goals, unknown effects and risky conditions, decision-making is a difficult and risky task. Risks can be minimized if the decisions are made systematically. In an earlier paper, we introduced methods of Decision Theory to perform such decisions in a rational way. This paper introduces a method for evaluating alternatives of architectural decisions, for both architectural design and refactoring. This method adopts elements of the scenario-based evaluation method ALMA [1]. A practical example illustrates the application of the improved decision process.

Keywords*: Refactoring, Reengineering, Software Architecture, Impact Analysis, Decision Theory*

1. Introduction

The complexity of software-intensive systems is increasing rapidly. Many of them are critical for the existence of an enterprise. If such a system could not adapted to changed market needs, this would lead to a loss of market share. Missing flexibility and maintainability represent high risks for mission-critical software. Some of the architectural decisions during the development and evolution of systems of this kind are frequently called strategic ones, because they determine future decisions and measures. Examples for such decisions are the definition of a middleware platform, of a database management system (DBMS) or an architectural style for distribution and cooperation between subsystems. Such decisions bear a high risk, because later changes require high effort and – even more important – delays. Unfortunately, these decisions are hard to make:

- The objectives and conditions are mostly complex, partly contradicting, and not completely known;
- Solution alternatives are hard to evaluate, because their consequences are mostly uncertain.

To reduce risks and to facilitate such decisions, a methodical and rational procedure was developed [2], based on the Decision Theory. Additionally, Impact Analysis is required to forecast the consequences of an alternative e.g., quality improvements, side effects and implementation effort. The identification of the consequences of a particular change in the software engineering domain is called Impact Analysis [4],[5]. Even if such a way of decision-making requires some effort, it increases the overall efficiency of the development processes by reducing risks. As a result, architectural decisions are made less subjectively.

The formerly developed procedure of rational decision-making [2] includes the following phases, as shown in fig. 2: identification of objectives, description of the architecture, and selection, analysis and evaluation of the alternatives. The benefit of this process results especially from the reduction of ambiguity, uncertainty and fuzziness of objectives and conditions, and from the way of weighting the consequences related to a broad variety of aspects.

Even if made in a rational way, the decisions are based on a proper establishment, selection and evaluation of architectural alternatives. Especially the Impact Analysis has to be done in a systematic way, because it is highly complex: Accurate positive *and* negative quality impacts have to be determined, and various side effects and conditions have to be considered, e.g., restrictions resulting from previous decisions. Furthermore, the Impact Analysis bears an uncertainty because it is related to a future – currently not existing – architecture, which has to be managed. The Decision Theory as the basis of the decision process does not provide methodical support for the Impact Analysis of architectural decisions, it just gathers the results.

Analyzing the Impact of alternatives, based on an evaluation of several executable prototypes of the resulting software system is too effort and time consuming for most cases. Estimations would require less effort, but pure subjective estimations bear the risk of

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

being vague, too optimistic or too pessimistic. Wrong or unrealistic estimations would lead to wrong decisions. However, additional effort for an Impact Analysis is justified if it is related to the risk of wrong decisions. If a wrong decision for a business critical system would lead to an organizational-wide delay of new products and services, e.g., if the distribution of a product is impossible for some days, it would have been much better to spend one or two days in making the best decision.

The goal of this paper is to extend the decision process on architectural alternatives by a scenario-based Impact Analysis. This Impact Analysis is developed with strong similarities to scenario-based methods of architectural assessment, especially ALMA [1]. Since ALMA was developed with different goals, it could not simply be integrated.

This paper is organized in the following way: At first, three major State of the Art approaches for Impact Analysis are compared and evaluated in section 2. In section 3, the decision process [2] is explained in brief. In section 4, the case study is introduced, which is applied in section 5 to illustrate the introduction of the an Impact Analysis to the decision process in detail, by showing its activities in a refactoring decision. In the conclusion, evaluation results and open issues are discussed.

2. State of the Art in Impact Analysis Approaches

The Impact Analysis was introduced to the software engineering domain because side effects related to a change constitute highly critical issues [4][5]. Side effects can occur in unchanged components, which are related to the changed ones, e.g., by dependencies, constraints or import-export relations. An Impact Analysis can be done before or after an implementation of a change.

Any Impact Analysis approaches have to reduce uncertainty and risks of the architectural decisions. In the background of this paper, the different approaches are therefore compared according to the following criteria:

Degree of reduction of uncertainty and risk: There are different way to reduce uncertainty and risks related to the design process, e.g., a prevention of subjective influences and uncertainty, as well as a systematic, rational procedure, an explicit manifestation of dependencies and constraints in models, and an analyses of dependencies such as side-effects.

Ability to forecast quality improvements, effort and functional changes: Quality properties of an software architecture are usually regarded as more important than functional properties. An Impact Analysis has to evaluate quality properties e.g., maintainability, portability, robustness, performance and scalability. Furthermore, the implementation effort and the impact on the functional behavior are important criteria.

Ability to manage the complexity of architectural decisions: Software systems and their architectures are highly complex. They are characterized by strong interrelations and constraints. Decision-making by humans requires for abstractions and simplifications to master this complexity.

Appropriate for usual development processes: The Impact Analysis has to fit into the decision process [2] as well as into development processes seamlessly.

Additional effort: The extra effort of the decision-making – including the one of the Impact Analysis – should be as low as possible. However, the higher the risk of a decision, the more effort is justified.

Degree of tool support and automation: Tool support and automation of the Impact Analysis would reduce its extra effort and would increase the efficiency of software and system development.

The following three major Impact Analysis approaches are investigated, described, and analyzed in the following section, whether they fulfill the criteria or not.

Graph-Based Analysis

The impact of architectural changes to a system is caused by dependencies between its parts. To describe an architecture formally, a graph-based model represents one option. In a graph model, the system's components and dependencies are expressed by nodes and edges. Graph analyses [4] can then be applied to investigate the effects of an architectural change: the more nodes are effected by a change, the higher is the effort. Path analyses can be applied to identify potential effects to the functional behavior. The establishment as well as the analysis of a graph-based model can be performed in a rather systematic way, thus the risks of subjectivity and uncertainty do not apply. However, most quality properties such as maintainability, reliability and robustness of a system are hardly to model within the same graphs representing the structure of that system, therefore this approach is not suitable to all relevant criteria. Performance can be mentioned as a non-functional criterion which can be evaluated well using graph-based analyses.

The formal description of a complete architecture including the architectural alternatives requires a much higher effort than a semi-formal one as usual for documentation purposes, because the complete system in-

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

cluding all details of its parts and its environment has to be described formally to enable an analysis. Graph-based analyses offer some potential for integration into usual development processes because they are based on models which could be established within the design phase. Their application requires special skills. Tool support is available for performing analyses, however the establishment of a graph model has to be performed manually.

Graph-based analyses are to some extend suitable to perform an Impact Analysis for architectural decisions, however they cannot be applied for most criteria as well as for low effort analyses.

Simulation

The simulation of the runtime behavior represents another method to analyze the consequences of architectural changes. Most simulation approaches focus on events, which are triggered by rules, and the traces of their effects.

To simulate an architecture, the relevant parts of the architecture, e.g., subsystems or components, are modeled by an Architecture Description Language (ADL), e.g., Rapide [16]. Components, interfaces and dependencies are modeled to express the structure of a system. Additionally, events, rules and trigger have to be defined to enable a simulation of the runtime behavior of the architecture after a change. The events are triggered through the defined rules and a tool captures and traces the effects [16]. If all relevant parts of an architecture are described, the resulting model can be executed to perform simulations.

To analyze architectural changes, the model has to be evolved prototypically according to the necessary changes. Establishing and evolving such architectural models requires a high effort, because the both the model and the prototype have to be modeled up to deep details. As a result, the developer can check if the events are processed correctly. Additionally, the impact on the performance can be analyzed.

The simulation is useful, if the performance of the software system is the most critical aspect of an architectural decision. Furthermore, the simulation eases the analysis if the functional behavior of the software system before and after changes. However, a concept is missing to analyze the quality-related consequences, like the impact on the portability or maintenance, in a systematic way.

Similar to graph-based analyses investigated above, simulation approaches offer some potential for integration into usual development processes because based on models. Furthermore, their application requires special skills, especially for establishing the models. Tool support is available for performing the simulations, however the establishment of the models requires high manual effort.

This high effort is only justified, if the simulation does help to solve very high risks concerning correctness and performance. The approach is less suitable if other non-functional criteria are more important.

Scenario-Based Analysis

A scenario-based architectural analysis consists of a formal review for the assessing a software architecture. The analysis is performed by developers in the way of a walkthrough following to a scenario, which is a peer-review following to that scenario. The Architecture Tradeoff Analysis Method (ATAM) [6] is one of the most popular methods, offering advantages in comparison to other scenario-based analysis methods [7]. ATAM describes a complete process of preparing and performing the walkthrough (see fig. 1). According to the relevant evaluation criteria, a scenario has to be established. For evaluating the correctness, a use case scenario is applied. Maintainability can be checked by following a change scenario. The selection of a scenario represents the most critical issue, because a wrong scenario does not help to discover deficiencies.

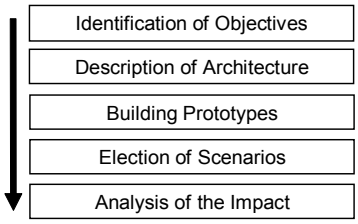


Figure 1. Phases of the scenario-based analysis by ATAM

The Modifiability Analysis Method ALMA [1] was developed based on ATAM, because ATAM is a general evaluation method, and an Impact Analysis requires specialized scenarios. ALMA includes some pre-defined scenarios for an evaluation regarding modifiability and therefore it can be applied here.

By performing the defined procedure by humans, a walkthrough is less objective than a graph-based analysis or a simulation. As a consequence, there are only limited opportunities for tool support and automation. To reduce the subjectivity, it has to be decreased by a proper organization of the team interaction. The effort can be reduced by investigating only the relevant parts of an architecture, affected by a change. However, the changes of concern have to be implemented to some extend to provide a basis for the walkthrough. In terms of effort reduction, only prototypical implementations are developed. In contrast to the aforementioned approaches, the developers can deal with syntactically and semantically incomplete models thus

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

reducing the modeling effort prior to the Impact Analysis.

As an advantage in comparison to the aforementioned approaches, all required quality criteria can be assessed by applying proper scenarios.

According to the risk level, analyses can be performed in a more or less sophisticated way, thus increasing or decreasing the required effort correspondingly.

The scenario-based analysis is able to manage the complexity of architectural decisions in a good way, because abstraction, hierarchical structuring and views can be used both in modeling and in performing the walkthrough. The approach fits well into usual development processes, because the establishment of prototypes can be performed within the iteration cycles.

It can be stated that a scenario-based approach is suitable in the best way for analyzing the impact of alternatives within an architectural decision process. In section 5, some central elements of the ALMA method will be applied in our process.

3.The Decision Theory Based Process

Decision-making for architectural development and reengineering is unable to reduce risks of mission-critical projects if performed in a subjective way and without a methodic procedure. To reduce the risks and to facilitate such decisions, a methodical and rational procedure was developed [2], based on the Decision Theory. Decision Theory facilitates a systematic and rational decision making, especially if competing objectives, side effects and uncertainty have to be considered [8]. Such decision making is broadly applied to solve economic decision problems, e.g., for financial investments. To improve the comprehension of the reader, an overview over the extended decision process is given here. Fig. 2 shows the overall structure of the process. In this section, the phases are explained in brief, and the introduction of the Impact analysis to phase 3 is explained in the next section. Only this phase is affected by the extension presented in this paper.

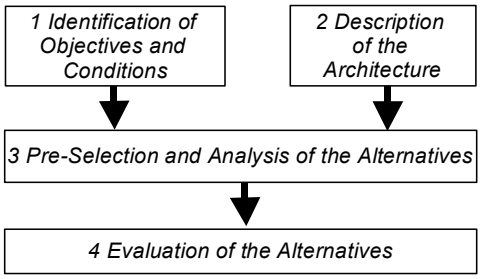


Figure 2: Phases of the Decision Process [2]

Identification of Objectives and Conditions

At first, all objectives and conditions relevant for a decision have to be identified. They are the central criteria for the whole architectural decision. In the case of competing, uncertain or fuzzy objectives this phase is a critical one.

The objectives are derived from the yet unsatisfied non-functional and functional requirements, which are relevant for an architectural artifact, e.g., a subsystem or a component. In the frequent cases of being incomplete, vague, inconsistent or competing, the objectives are refined and classified into two categories: mean objectives and fundamental ones. The fundamental objectives are the primary ones; they bear a high value for the stakeholder. Means objectives are those contributing to the fundamental ones; their results help to achieve the fundamental objectives. In addition to the classification, the objectives are structured and prioritized. The priorities help to solve conflicts. Furthermore, the priorities are applied for reducing the number of objectives under review, to reduce the overall effort for a decision.

In addition to the objectives, the conditions have to be identified. There are three types of conditions: organizational and technical ones as well as conditions caused to previous decisions. Organizational conditions are defined by an enterprise or by a customer, e.g., enterprise strategies, know-how of the staff, deadlines, budget. Usually they are hard to change. Technical conditions are represent the technical environment a product is developed for or developed in, e.g., standards, tools and protocols. These conditions tend to be variable over time. Conditions of both types are usually identified using checklists. The third type of conditions result from previously made operational or strategic decisions and from their implementation. Both objectives and conditions are in phase 3 applied as the criteria for evaluating of the architectural decisions.

Description of the Architecture

In the second phase, a description of the current state of the software architecture in form of an architectural model is developed. This model serves as the basis for establishing prototypes for the later evaluation of the alternatives. The model includes all parts of the architecture, which are relevant for the actual architectural decision, like components, interfaces and relations. For example, if improving maintainability of the database access is an objective, then all parts of the architecture have to be described, which are related to the database access. The more risky a decision, the more parts of the architecture are important enough to be described. To reduce uncertainty and risks, the de-

scribed model has to be consistent and clear to understand – with respect to the criteria relevant to a certain decision. Usually, an architectural model is separated into views and described by a set of diagrams, in some cases using an Architectural Description Language (e.g., ACME [9], xADL [10]), or UML.

Preselection and Analysis of the Alternatives

In the third phase of a decision, suitable alternatives are established and their consequences are analyzed. To reduce the effort of analysis and evaluation, the number of alternatives under review is reduced stepwise, while consulting the objectives, the conditions and the risks.

The alternatives for architectural decisions are established based mostly on the experience of the architect. They stem from a variety sources of two main categories: abstract and technological solutions. Abstract solutions as architectural and design patterns, architectural styles or heuristics belong to the first category. These solutions have to be instantiated and implemented to provide a solution and to achieve the objectives.

Technological solutions as the second category consists of reusable building blocks, solution parts from previous projects, third-party components, frameworks as well as technological standards, e.g., component models like those of COM or CORBA.

For each suitable alternative, the consequences according to the objectives and conditions have to be determined. Since the alternatives are not yet implemented at this time, it is only possible to establish a hypothesis or estimate the consequences of each alternative. The decision process [2] suggests a preselection of the alternatives to reduce the effort, and then a detailed estimation of the consequences for the preselected ones. The preselection is based on one hand on the properties and the general quality characteristics of the alternatives. If for example the Design Pattern Façade [14] is an alternative, it influences the architectural quality by increasing the maintainability, because the encapsulation of elements is improved. On the other hand, the alternatives are compared to the organizational and technical conditions. If there is no match, an alternative cannot be applied.

If performed subjectively by the architect, this selection and the estimations are vague and risky. For high-risk decisions, a more detailed analysis is necessary to identify the consequences and the benefits or disadvantages. As a contribution of this paper, this phase is improved by introducing a scenario-based Impact Analysis, as explained in section 4.

Evaluation of the Alternatives and Decision

The fourth and last phase of the decision process is an evaluation of the alternatives by a procedure adapted from the Decision Theory [15]. The several consequences are aggregated and weighted to enable a rational decision. In this way, the risk of subjective decisions is reduced, e.g., if the developer could focus more on positive consequences and would not consider the negative ones.

The evaluation consists in the following steps:

Normalization: In the first step the consequences according to different properties are normalized. A comparison of the consequences is facilitated in this way even if they are of different scales. The consequences are normalized to an interval between zero to one. The best consequences are normalized to one, the worst ones are normalized to zero. If there are only minor differences between consequences, some realistic values for the best and the worst values have to be estimated.

Weighting: In a second step the consequences are weighted according to the relevance of the different properties. Preferred properties are weighted higher.

Calculation of the Expected Value: This third step is performed to take probabilities of the consequences into account. For example, if there is a chance that an alternative requires additional effort, or if there is a risk that a wanted property is not achieved, the probabilities of these different consequences are multiplied with the weighted values to calculate an expected value.

The developer can then select the alternative with the highest value.

4. Typo3 Refactoring Case Study

For illustrating the Impact Analysis extension, a case study is now introduced in brief . The extended method is then described in section 5 using examples form this section.

As case study, a refactoring decision on the Typo3 system is applied. Typo3 is a popular, open source Content Management System for editing and publishing web pages. An author creates web pages by a simple and easy-to-use editor, and the created web page content stored in a MySQL database. If a client requests a web page, the content is retrieved from the database and the webpage is dynamically generated (see fig. 3). Inside the Content Management System Typo3, the Core provides the basic functions to manage the content and to generate the web pages. These basic functions are extended by plug-ins - so-called extensions - e.g. to integrate multimedia elements into the webpage. To store and retrieve the content from the

database, a Database Abstraction Layer provides the required functions. Typo3 is a mission critical software system for web-based organizations, which have to share information in a very flexible way, like web-based customer operations.

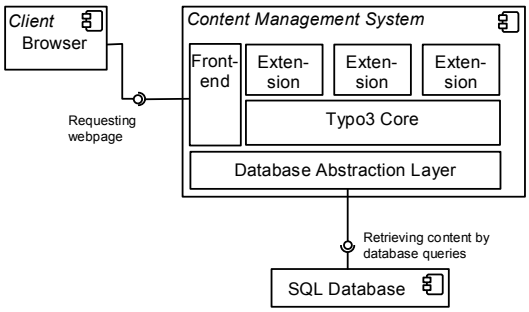


Figure 3. Overview over the Typo3 Architectural Model

4.1. Objectives and Conditions

For this case study decision, four objectives and conditions have to be considered: *portability, performance, minimal implementation effort and avoidance of functional changes*.

For Typo3 version 3.8.1, the *portability* is limited because Typo3 does not support other database management systems (DBMS) than MySQL. This fact limits its applicability if the MySQL DBMS cannot be used because of security reasons or other restrictions. For improving the portability with respect to the DBMS a refactoring has to be performed. The main issue of this refactoring consists in a change of the architecture, so that Typo3 is able to cooperate with other DBMSs. This refactoring bears important risks because changes to the DBMS access affects the whole Typo3 functionality of storing and retrieving content. A detailed analysis of the consequences of the alternatives is necessary for a rational decision-making. Therefore, the extended decision process is applied here. The justification of the additional effort of the Impact Analysis is derived from the risk of failures and unconcerned side effects.

Performance of the DBMS access represent another objective. Usually performance and portability are competing objectives because they demand for contradicting solutions e.g. optimization of data transformation versus layering and encapsulation. Performance is a critical objective, because the storage of the content and the generation of web pages require a lot of database queries, and missing performance of the database access would lead to a delays of providing requested web pages. This would affect the usability and reduce acceptance of the web service and thus of Typo3. Therefore, the time to process database queries must not be lengthened.

4.2. Refactoring Alternatives

Among several possible alternatives available for a restructuring of the database access, the following two alternatives are suitable (see fig. 4 for an overview):

ADODB: The first alternative consists in the application of the ADODB library. ADODB provides functionality to transform SQL statements according to several DBMSs, e.g., to Oracle DBMSs. Even if ANSI SQL is a standard language, there are many proprietary extensions. For example, the data format of time stamps is not standardized and nearly every DBMS uses a different format. Therefore, some SQL statements require a transformation, which is done by the functionality of ADODB.

Wrapper: The second alternative consists in the development of several Wrappers, which are specialized to one DBMS. For example, if Typo3 has to work with the Oracle DBMS, an Oracle wrapper has to be developed and implemented. Such a wrapper includes native database functions, like ora_parse(<SQL-Statement>) for Oracle or the MySQL function, shown in the previous section. Due to space reasons, we decided to consider only two new Wrappers in our case study: one for Microsoft's SQL-Server and one for Oracle products.

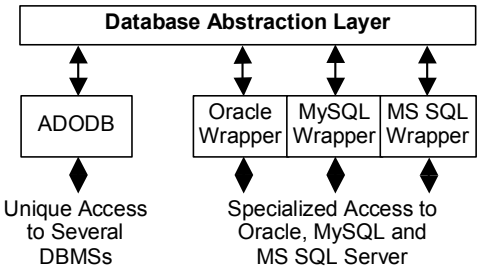


Figure 4. Access to different DBMSs by the two alternatives

The evaluation of the two alternatives within the decision process is now performed by the Impact Analysis in the next section.

5. Introducing Impact Analysis for Architectural Decisions

This section introduces a methodical Impact Analysis to the decision process described in section 3. The Impact Analysis extends phase 3 of the decision process (see fig. 2). It performs an analysis of the formerly established alternatives, according to the objectives and conditions from phase 1 of the process, and with respect to the architecture as described in phase 2. By

the Impact Analysis, the consequences of the alternatives are determined, which are the basis for the evaluation and decision in the last phase.

The scenario-based approach ALMA was evaluated best in the State of the Art analysis in section 2; only for pure performance analyses a simulation-based approach would be appropriate as well.

ALMA was developed mainly for assessing complete architectures, but here only alternatives for architectural decisions have to be assessed. Therefore only a few core elements of ALMA have to be applied. The analysis of objectives e.g., is already covered by phase 1 of the process (fig. 2). For effort reduction, only a scenario-guided walkthrough of prototypical architectural models is performed. If performance is among the most critical objectives, the architectural models are developed in a way that enables a simulation.

5.1. Building Prototype Models

An Impact Analysis by a scenario-based walkthrough requires an object to investigate. To minimize the implementation effort, alternatives are implemented to certain extend depending on the risk.

Only in the case of high-risk objectives and conditions, alternatives are implemented by evolving the architectural model of phase 2 (see fig. 2); the result is here called a prototype model. To reduce this effort, the prototype models are only developed as necessary for an assessment according to the objectives and conditions of concern. Furthermore, only the few (2 or 3) of the preselected alternatives are developed into prototype models.

In the case of low risks, only estimations are performed.

Typo3 Case Study:

The preselected alternatives interoperate with the Typo3 Core through the database abstraction layer (see fig. 4). The SQL statements are transformed by one of the two alternatives explained in section 4. Two prototype models are necessary, built by the following activities (see fig. 5):

ADODB alternative: The first step is the implementation of the ADODB library, which can be done as a system extension. The second step are adjustments to the database abstraction layer, so that the SQL commands are rerouted to the ADODB for transformation.

Wrapper alternative: The first both steps are the development of the wrapper for the Oracle and the MS SQL-Server DBMS. The third step is an adjustment of the database abstraction layer in a way, that the SQL commands are directed through the DBMS-specific Wrappers.

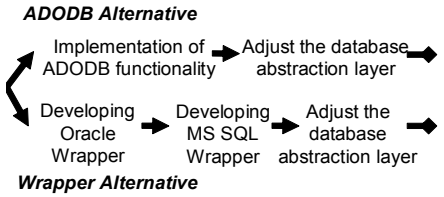


Figure 5. Decision Tree of the implementation activities

5.2. Selection of the Scenarios

According to ALMA, the determination and selection of proper scenarios is performed according to the objectives and conditions. Four scenarios were established:

Criterion	Scenario
Flexibility, modifiability, effort	Change scenarios developed for ALMA
Reliability	Introduction of failures
Interoperation	Typical use cases
Performance, resource consumption	Stress scenario: high volume of requests or data

The set of scenarios is then customized according to the precise objectives and conditions assigned to that particular decision. Furthermore, the scenarios are modified in order to reduce the effort for building the prototype model, e.g. by reducing the scope of a scenario.

Typo3 Case Study:

According to the objectives and conditions, the following scenarios are developed:

Portability: To evaluate the alternatives with respect to portability, a migration scenario is established. The scenario analysis then focuses on the effort, which is required to migrate Typo3 to another DBMS, e.g., FoxPro.

Performance: To check, if the performance has changed, some stress scenarios regarding database access are established. These scenarios are developed from the use case "request webpage" with a frequency of 90 % of the original Typo3 stress threshold.

5.3. Application of the Scenarios and Analysis

The scenarios are now applied to the prototype models of the two alternative. According to the scenario-based approach, this step is performed by walkthroughs.

If performance is among the most critical objectives and the risk level justifies extra effort, the behavior of the prototype model can assessed by a simulation much better than by a walkthrough. To enable a simulation, the prototype model has to be transformed into a proper ADL, e.g., Rapide [16].

Identification of Quality-Related Consequences

For the identification of quality-related consequences, the impact of conditions resulting from the alternative under review has to be considered. For the later decision both, the degree and the probability of the fulfillment of an objective or condition, have to be identified.

Quality-related consequences of an architectural alternative can be determined by considering the positive and negative impact, introduced by properties and the side-effects. They can improve or decay the extend to which objectives are achieved. Major sources for side-effects and properties are the technical conditions to be identified in phase 1 (see fig. 2), resulting e.g., from formerly implemented patterns, styles or used protocols. Each of the identified technical conditions has to be checked, if there is an impact on the objectives. An example is an implementation of the Proxy design pattern [14] to improve the performance of an access to a remote component. A restrictive layer-architecture can hinder these performance improvements.

In the case of a negative impact, the alternatives have to be extended by additional activities to reduce the unwanted impact and to achieve the objectives. If the occurrence of an impact is uncertain, these additional activities are optional. In the latter case, the implementation of the alternative varies, either with or without the optional activities. The probability of the occurrence will later be used to the consequences. The estimation of the probabilities is based both on the experience of the developer and on methods of the Decision Theory [15]. In the case of the Proxy pattern example described above, it could be uncertain, if the layered architecture indeed has an negative impact. Therefore, the implementation of a bridge over the layers would be an optional activity with a certain probability.

Typo3 Case Study:

The identification of the quality-related consequences is determined by applying the scenarios to the prototype models in walkthroughs, as mentioned in section 5.2. For the **ADODB** alternative, the walkthrough shows that the portability is improved, because the flexibility is improved regarding all DBMSs which are supported by ADODB. Additionally, in the walkthrough it was stated that there is a negative impact on the maintainability: future extensions to the DBMS' functionality are not utilizable if they exceed the functionality provided by the ADODB adapter. Furthermore, the possibilities of bug tracking are considered to be reduced, because it is difficult to locate errors. Performance problems are expected due to the additional transformations to SQL commands and due to the fact, that some advanced database functions, e.g., caching mechanisms cannot be used.

For the **wrapper** alternative, the same scenarios are applied. The walkthrough shows that the portability is improved. Similarly to the ADODB alternative, the bug tracking was considered to be limited. As an advantage over the latter it is stated that the DBMS-specific wrappers make advanced database functions available and that additional transformation steps are not required. Therefore the performance should not affected, compared to the unchanged Typo3.

The analysis results of the quality-related consequences are shown in fig. 6. Grades from 1 (best) to 5 (worst) are used to quantify the extent the objectives are achieved. The consequences of the ADODB alternative are evaluated for portability to grade 1 and for performance to grade 3. The consequences of the wrapper alternative are evaluated both for portability and performance to grade 1.

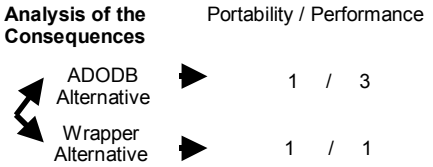


Figure 6. Quality-Related Consequences as Decision Tree

Identification of Functional Changes

Furthermore, the functional behavior of the future architecture has to be checked, either if it remains unchanged in the case of refactoring or if the desired behavior is achieved in the other cases. These checks are performed by walkthroughs based on use case scenarios. If there are unwanted or missing functional changes, the alternative has to be modified.

Typo3 Case Study:

As typical for refactoring activities, changes to functional behavior are unwanted. For the case study no additional scenarios are applied, but during the walkthrough mentioned above the issue of functional changes is analyzed. As typically for adapters, the **ADODB** alternative hides special database functions from use, for example the searching algorithms of MySQL. As a consequence, some search operators, e.g., NEAR, cannot be used. This leads to a change of the functional behavior. To make such functions to operate, additional refactoring steps have to be performed. Potentially, there are more specific database functions, which have to be supported. However, exploring this issue would require a detailed source code

analysis, which is not covered by this architectural Impact Analysis itself. The developer estimates the probability of these additional steps by 40% (see fig. 7). The quality-related consequences would not be affected if such additional steps would be performed.

The **wrapper** alternative is functionally comparable to the original database access in Typo3 version 3.8.1. If there are no implementation errors of the wrappers, the functional behavior should be unaffected.

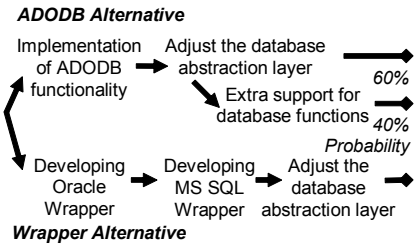


Figure 7. Activities and Consequences Concerning the Functional Behavior as a Decision Tree

Estimation of the Implementation Effort

For effort as a critical condition, an effort estimation is carried out during the third phase of the decision process. Efforts for the later implementation of an alternative, for changes and for a necessary modification of the rest of the architecture can be performed by developers during the elaboration of the prototype model and during the walkthrough.

Typo3 Case Study:

The **ADODB** alternative requires are two major and one optional development activity. Furthermore, version conflicts with the remaining development activities of the Typo3 core will likely occur and have to be managed. An effort of 35 person-days is estimated. In the case that specific database functions have to be supported, the effort rises to 45 person-days.

The implementation effort of the **wrapper** alternative is higher, because each wrapper has to be developed and tested separately. Similarly to the other alternative, version conflicts will likely occur. An effort of 50 person-days is estimated.

5.4.Typo3 Case Study Results: Evaluation of the Alternatives

The Impact Analysis provides the required data for the evaluation of the alternatives. To conclude the case study, the succeeding steps of normalization, weighting and calculating of the expected value is described here briefly.

Normalization: The best alternative requires an effort of 35 person-days, the value of the corresponding consequence is normalized to 1. The best alternatives concerning portability and performance are of grade 1, their corresponding consequences get value 1. The worst alternative concerning effort requires 50 person-days, its consequence is normalized to 0. The worst alternative concerning performance is of grade 3, but an even more worse performance of grad 5 is possible. Therefore, the consequence is normalized to the value 0.5. In the case of the additional refactoring activities of the ADODB alternative, the effort rises to 45 person-days. This effort is near the worst consequence, so we normalize it to 0.3.

Weighting: In our case, a high portability and a low implementation effort are most important. Therefore, the weighting factors are determined to 0.4 for the portability and 0.3 for the effort since Typo3 is not a real-time application but performance is important, the weighting factor is defined to be 0.3 for the performance. The normalized consequences (see fig. 8) are multiplied by these weighting factors and added together. The total value of the ADODB alternative is 0.85 in the first case without additional refactoring activities. In the second case with extra activities for refactoring additional database functions, the resulting total value is 0.64. The total value of the wrapper alternative is 0.70.

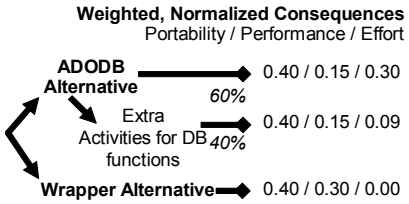


Figure 8. The Weighted Consequences

Calculation of the Expected Value: The consequences of the first variant of the ADODB alternative are multiplied with its probability of 0.6; the consequences of the second one are multiplied with 0.4. The results are added to get the expected value of 0.77. for the wrapper alternative there are no probabilities.

Finally, both alternatives are evaluated. The ADODB alternative gets a total value of 0.77, the wrapper alternative one of 0.70, thus there is only a minimal difference. The rational decision consists in a preference of the ADODB alternative against the wrapper.

6. Summary and Conclusion

In this paper, an architectural decision process is extended by a methodical Impact Analysis, in order to reduce risks by more systematic and confident decisions . The architectural decision process was previ-

ously improved by introducing elements of decision theory [2]. Risk reduction is achieved by methodically guiding the decision maker in the evaluation of the consequences of alternatives of an decision. For this evaluation, a scenario-based approach has been chosen to analyze the consequences of the alternatives according to systems behavior, quality and effort. This scenario-based Impact Analysis adopts elements of the ALMA method for architecture assessment. The phases of the extended decision are described in detail and the application of process is shown by a refactoring case study.

The application of the scenario-based Impact Analysis facilitates the decision-making regarding the architectural alternatives by evaluating the consequences of each alternative in a methodical, rational way. Without this Impact Analysis, no guideline for performing evaluations about the consequences was given, with many cases of subjective evaluations and remaining uncertainty as a result. By the introducing of the Impact Analysis step, the risk of side effects, additional effort and a changed behaviour is significantly decreased.

Nevertheless, the application of the Impact Analysis as well as the whole decision process introduces some additional effort. Therefore, the risk of each architectural decision has to be investigated if it justifies that effort. Examples for risks are missing flexibility of a system with increased time to market and effort, missing integration and collaboration opportunities to neighbouring systems, tremendous additional effort for later refactoring due to wrong basic architectural decisions. The issue of effort is tackled by suggesting different levels of analysis and decision effort according to the level of risk.

7.Acknowledgements

We wish to thank our colleagues Alexander Pacholik and Patrick Maeder for their helpful comments to an earlier version of this paper.

8. References

[1] P. Bengtsson, N. Lassing, J. Bosch and H.v. Vliet, *Architecture-level modifiability analysis (ALMA)*, The Journal of Systems and Software, Vol. 69, 2004, pp. 129-147.

[2] S. Wohlfarth and M. Riebisch, *Evaluating Alternatives for Architecture-Oriented Refactoring*, Proceedings of the 13th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS2006), IEEE Press, Los Alamitos (CA), 2006, pp. 73-79.

[3] M. Fowler, K. Beck and E. Gamma, *Refactoring: Improving the design of existing code*, Addison-Wesley, Boston, 2005.

[4] Robert S. Arnold and S.A. Bohner, *Software Change Impact Analysis*, IEEE Computer Society Press, Los Alamitos, CA, 1996

[5] S.L. Pfleeger and S.A. Bohner, *A Framework for Software Maintenance Metrics*, IEEE Transactions on Software Engineering, 1990, pp. 320-327.

[6] R. Kazman, M. Klein and P. Clements, *ATAM: Method for Architecture Evaluation*, Technical Report CMU/SEI-2000-TR-004 ESC-TR-2000-004, http://www.sei.cmu.edu/pub/documents/00.reports/pdf/00tr004.pdf, 2000.

[7] M.A. Babar, I. Gorton, *Comparison of Scenario-Based Software Architecture Evaluation Methods*, 11th Asia-Pacific Software Engineering Conference (APSEC'04), 2004, pp. 600-607

[8] E. Forman and M.A. Selly, *Decision By Objectives*, World Scientific Publishing Company, 2002.

[9] D. Garlan, R.T. Monroe und D. Wile, *Acme: Architectural Description of Component-Based Systems*, In: Gary T. Leavens und M. Sitaraman (eds.), *Foundations of component-based systems*, Cambridge University Press, New York, pp. 47 – 67.

[10] E.M. Dashofy, A.V.d. Hoek, R.N. Taylor, *A Highly-Extensible, XML-Based Architecture Description Language*, Proceedings of the IEEE/IFIP Working Conference on Software Architecture (WICSA'01), IEEE Computer Society, pp. 103 – 112.

[11] C. Hofmeister, R. Nord and D. Soni, *Applied software architecture*, Addison-Wesley, Reading (Mass), 2000.

[12] N. Medvidovic, *ADLs and Dynamic Architecture Changes*, Joint Proceedings of the SIGSOFT'96 Workshops, ACM Press, 1996, pp. 24-27.

[13] J. Bosch, *Design & Use of Software Architectures: Adopting and evolving a product line approach*, Addison-Wesley, Harlow, 2000.

[14] E. Gamma, R. Helm, R. Johnson and J. Vlissides: *Design Patterns – Elements of reusable object oriented Software*, Addison-Wesley, Munich, 2004.

[15] R. Clemen and T. Reilly, *Making Hard Decisions with Decision Tools*, Pacific Grove, Brooks Cole, 2004.

[16] Rapide Design Team: *Guide to the Rapide 1.0 - Language Reference Manuals*, URL: 'http://pavg.stanford.edu/rapide/lrms/overview.ps', last visited 2006-09-10.

Reengineering a Legacy Tool for Software Evolution

Chia-Chu Chiang

Department of Computer Science
University of Arkansas at Little Rock
2801 South University Avenue
Little Rock, Arkansas 72204-1099, USA
E-mail: cxchiang@ualr.edu

Abstract

This paper describes how a commercial legacy tool originally developed for a centralized computing environment has been reengineered for a heterogeneous distributed computing environment which allows the tool to be accessed remotely. In addition, we discuss how the tool was further enhanced to componentize customers' business applications in a heterogeneous distributed computing environment.

Keywords: Components, CORBA, Distributed Objects, Legacy Integration, Middleware, Wrapper

1. Introduction

As a software vendor, we developed a tool to help customers maintain their business applications. The tool was originally developed for a centralized architecture and had IBM mainframe hosts only. As information technologies evolve, computing environments become more heterogeneous and distributed. With the advances of current computing technology, our tool has already become a legacy tool. One of the issues was the tool could not be accessed remotely in a heterogeneous distributed computing environment. To achieve this goal, we did an improvement on this tool using the wrapping technique.

In Year 2000, we initiated a new project to preserve the values of the tool. One function the tool originally provided was to allow customers to extract the relevant code for testing using the program slicing technique [8]. In this project, we extended the program slicing technique to extract the relevant code corresponding to a business rule and then transform this code into a CORBA component in COBOL. The component can then be used across different business applications for reuse. The tool can be helpful in reengineering legacy business applications into component-based applications which can respond to the changes more efficiently than the traditional business applications [5].

The rest of this paper is organized as follows. In Section 2, a historical background of a centralized tool running on an IBM mainframe will be described. We then present a wrapper technique to integrate the centralized tool in a distributed computing environment which makes the tool available to be accessed remotely. In Section 4, we will describe a project called legacy modernization in which reusable business rules can be extracted out into CORBA components in COBOL. Through the CORBA communication infrastructure, components in COBOL can be invoked by remote clients in a heterogeneous computing environment. We summarize the paper in Section 5.

2. Historical background of the legacy tool

A comprehensive, integrated tool set [1] was developed to enable the maintenance of business applications. The tool enables detailed understanding of the existing business applications in a portfolio and automates traditionally time-consuming programming tasks. The tool is mainly used to reverse engineer the business applications running on IBM mainframes as shown in Figure 1.

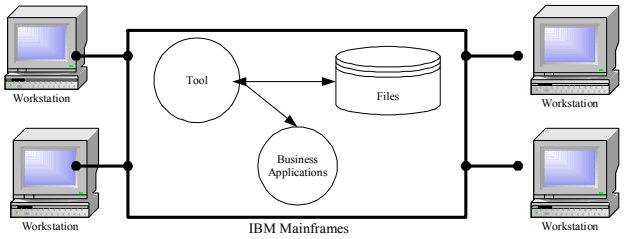


Figure 1. A centralized computing environment for business applications

The characteristics of the legacy tool are:

- It is domain specific.
- It uses an in-house developed database management system.
- It is a large, complex, and monolithic system.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

- It typically runs on IBM mainframes and has large, centralized in-house staffs for its development, maintenance, and customer support.

3. The wrapping approach

In order to make the tool to be remotely accessed, we built a wrapper around the tool using CORBA. With the wrapper, the tool becomes a reusable distributed component in a heterogeneous computing environment as shown in Figure 2. The tool can then be invoked from a remote client. CORBA provides the transparency of this access without regard for location or implementation of the tool.

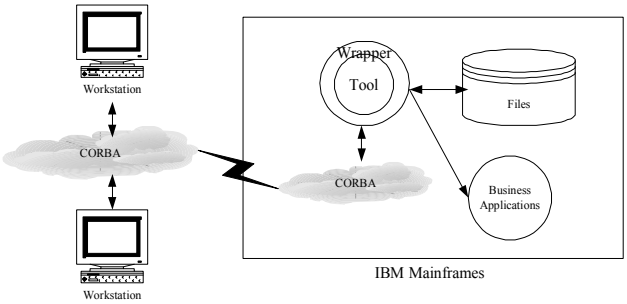


Figure 2. Legacy integration using wrapper

A client sends a request to the wrapper that will invoke the tool to analyze the customer's business applications in COBOL, JCL, CICS, DB2 SQL, IDMS, and IMS. The tool then writes the information about the programs/systems to a repository (files). The client can then send a request to the wrapper to query the results of the analysis. The legacy wrapping approach avoids rewriting or converting the tool, but has modernized the legacy tool to have the capability to cooperate with the customer's heterogeneous distributed computing environments. The approach enhances the product value and adapts the tool to new market needs. However, this approach is generally only a stopgap solution, rather than a strategic one. Nevertheless, from an economical point of view, the approach is a cost-effective way of modernizing the tool using middleware. The detailed implementations of the wrapper technique are discussed in the reference [2].

4. The reengineering approach

In Year 2000, we initiated a new project namely, legacy modernization in which business rules in legacy systems are extracted out from a business application into a CORBA component in COBOL. The reasons of reengineering a business application into components are listed as follows:

- For software reusability, customers want the reusable business rules to be shared by other applications so they don't have to duplicate the same business rules across the entire applications.
- Due to the availability of new technologies such as middleware and web services, customers want to pull their reusable business rules from their legacy code into components and prepare for e-business.

We define a primitive business rule as a basic function that takes inputs and produces only one output expressed in the equation: $O = F(I_1, I_2, ..., I_m)$. To be realistically applicable to large programs, we classify the program code into three categories – user interface, business logic, and data access. A set of primitive functions is collected together to implement a complex business rule as expressed in the equation: $(O_1, O_2, ..., O_n) = \cup_{i=1..n} F_i(I_1, I_2, ..., I_m)$. A component which implements related business rules is recovered by aggregating existing functions around a group of data without modifying the statements inside of those routines.

Locating a business rule is the starting point of aggregating relevant functions implementing the business rule. A program slicing technique using control flow graph, control dependence graph, and data dependence graph is primarily used to identify relevant functions that create or define the business rules.

Having collected all functions relevant to a business rule from programs; it is very common that each function relevant to a business rule may not exactly match in syntax and semantics. A user intervention is needed to analyze these functions and create a final one to be included in a component [7]. To convert the collected functions corresponding to a business rule into a component, a CORBA IDL [6] interface with relevant data structures is first created from these functions.

A server skeleton is then generated by the IDL compiler from the IDL interface. The functions as the process logic are added to the operations in the server skeleton. A complete server is next compiled, and linked with the CORBA runtime library and the COBOL Object Adapter (COA) library to create a server implementation. Figure 3 shows the activities for building a server in COBOL. The detailed implementations of a server can be referenced in [3].

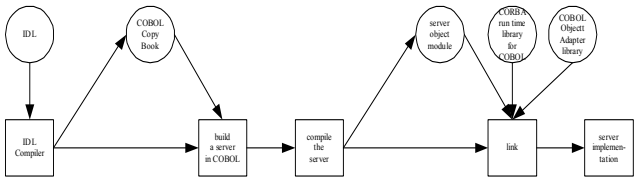


Figure 3. Process for creating servers in COBOL

A COBOL-based client can then be developed to invoke the operations on the server through the CORBA concepts of Dynamic Invocation Interface (DII). DII

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

allows the client to invoke the server's operations whose signature is not known at compile time and the server examines the signature of these operation calls and implements them at run time.

To support the resolution of the signature of operation calls at run time, the IDL interface is compiled into an interface description in COBOL as a copybook which will be included in the client and server code. As the client and server communicate with each other, they need to agree with the interface; otherwise, incorrect data and unexpected results may occur during the data transmission. Figure 4 presents an example of an IDL and its corresponding interface description in COBOL.

```
module SimpleHelloWorld
{
    interface GoodDay
    {
        string Hello();
    };
};
```

(a) An IDL

```
000010 01  SIMPLEHELLOWORLD-GOODDAY-OPERA PIC X(5) VALUE SPACES.
000020   88  SIMPLEHELLOWORLD-GOODDAY-HELLO VALUE "Hello".
000030 01  SIMPLEHELLOWORLD-GOODDAY-HE000.
000040   03  RESULT                    POINTER.
…
…
003910 01  INTERFACE-NAME-BLOCK.
003920   03  INTERFACE-LENGTH         PIC 9(10) BINARY VALUE 24.
003930   03  INTERFACE-STRING         PIC X(24).
003940     88  INTERFACE-SIMPLEHELLOWORLD-GOO
003950                VALUE "SimpleHelloWorld/GoodDay".
```

(b) Interface description

Figure 4. A sample IDL and the corresponding interface description in COBOL

The COA plays a key role in facilitating the communications among COBOL applications such as data exchange between a client and a server, invocation of ORB functions, and memory management. Figure 5 shows how the data are passed by the COAs interfacing with ORB between the client and server.

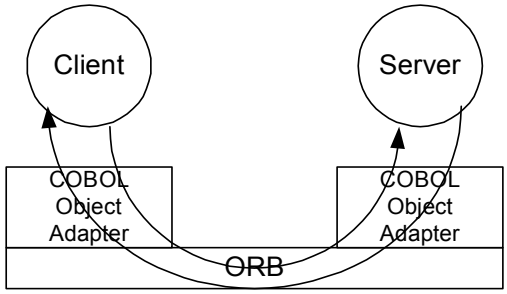


Figure 5. Data transmission between the client and server

To understand how a COBOL server works with a client, the following scenario in Figure 6 illustrates how the functions in the COA cooperate with the functions in the CORBA runtime library to form a functioning server program and a client program.

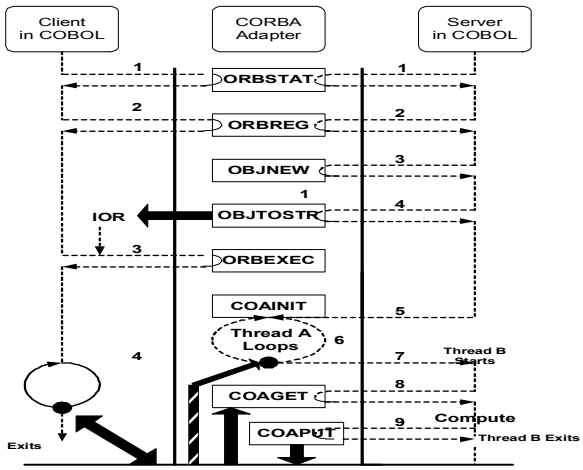


Figure 6. Invocations of functions of the COBOL object adapter

To provide services to the clients, a server needs to be started up first. ORBSTAT makes the status of successive calls available for the server. The server then calls ORBREG to register its interface to its COA, OBJNEW to create a CORBA object reference, and OBJTOSTR to convert the newly created object reference to a string. The string will be stored in an IOR file which will be read by the client for object invocation. The server then calls COAINIT to start listening for incoming requests. Control is transferred to ORB which enters an infinite loop waiting for client requests to arrive. A sample "hello" server program is shown in Figure 7.

```
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. HELLO.
000030 DATA DIVISION.
000040 WORKING-STORAGE SECTION.
000050 01  SERVER-NAME            PIC X(3) VALUE "hello".
000060 01  SERVER-NAME-LEN        PIC S9(9) BINARY VALUE 5.
…
000110* The next copy book is generated by the IDL compiler.
000120     COPY HELLO.
…
000160 PROCEDURE DIVISION.
000170 HELLO-INITIALIZATION.
000180     CALL "ORBSTAT" USING ORB-STATUS-INFORMATION.
000190     CALL "ORBREG"  USING SIMPLEHELLOWORLD-GOODDAY-INTER.
000200     CALL "COAINIT" USING SERVER-NAME, SERVER-NAME-LEN.
000210     GOBACK.
000220 ENTRY "DISPATCH".
000230 HELLO-DISPATCHER.
…
000290* Dispatcher for the interfaces
000300     EVALUATE TRUE
000310     WHEN INTERFACE-SIMPLEHELLOWORLD-GOO
000320         PERFORM DISPATCH-SIMPLEHELLOWORLD-GOOD
000330     END-EVALUATE.
000340     GOBACK.
000350* Operation Dispatcher for interface
000360 DISPATCH-SIMPLEHELLOWORLD-GOOD.
000370     CALL "STRGET"  USING OPERATION-NAME,
000380                    SIMPLEHELLOWORLD-GOODDAY-OP000,
000390                    SIMPLEHELLOWORLD-GOODDAY-OPERA.
000400     EVALUATE TRUE
000410     WHEN SIMPLEHELLOWORLD-GOODDAY-HELLO
000420         PERFORM DO-SIMPLEHELLOWORLD-GOODDAY-HE
000430     END-EVALUATE.
000440
000450 DO-SIMPLEHELLOWORLD-GOODDAY-HE.
000460*    Server logic for interface SimpleHelloWorld/GoodDay
000470*    operation Hello.
…
000500     CALL "COAPUT" USING SIMPLEHELLOWORLD-GOODDAY-HE000.
```

Figure 7. A sample "hello" server

Once the server is ready, a client can submit a request to the server by making the following sequence of calls: ORBSTAT, ORBREG, and ORBEXEC. ORBSTAT makes the status of successive calls available for the client. ORBREG allows the client to register the interface to its COA and then invokes ORBEXEC to call an operation on the server. A sample client program corresponding to the "hello" IDL is shown in Figure 8.

```
000010 IDENTIFICATION DIVISION.
000020 PROGRAM-ID. CLNCOB.
000030 ENVIRONMENT DIVISION.
000040 CONFIGURATION SECTION.
000050 SOURCE-COMPUTER. IBM-PC.
000060 OBJECT-COMPUTER. IBM-PC.
000070 DATA DIVISION.
000080 WORKING-STORAGE SECTION.
000090 COPY HELLO
…
000150 PROCEDURE DIVISION.
000160      CALL "ORBSTAT" USING ORB-STATUS-INFORMATION.
000170      CALL 'ORBREG' USING SIMPLEHELLOWORLD-GOODDAY-INTER.
000180      CALL 'IORTOOBJ' USING IOR-FILE-NAME
000190                            OBJECT-REFERENCE.
000200      SET SIMPLEHELLOWORLD-GOODDAY-HELLO TO TRUE.
000210      CALL 'ORBEXEC' USING OBJECT-REFERENCE
000220                           SIMPLEHELLOWORLD-GOODDAY-OPER
000230                           SIMPLEHELLOWORLD-GOODDAY-HE00O
000240                           GOODDAY-USER-EXCEPTIONS.
…
000280      DISPLAY 'RESULT:' RESULT.
000300      GOBACK.
```

Figure 8. A sample "hello" client

For each client's operation call, the client's COA and the server's COA exchange data according to the signature of the operation described in *InterfaceDescription*. The client's COA retrieves the data from the client, stores the data in its address space and passes the data to the server's COA which will store the data in its address space. The server then invokes COAGET to copy the data from its COA's address space to its program variables for processing. If there are output data to be returned to the client, the server invokes COAPUT to pass the data to its COA which will forward the data to the client's COA. The client's COA copies the output to the client's program variables. In the "hello" IDL, since there is no input to be passed to the server, COAGET is not needed in the server program. The detailed implementations of the COA are discussed in [4].

5. Summary

Legacy systems were mostly developed for centralized environments. These systems are usually inflexible and non-scalable. In addition, with the advent and widespread use of object-oriented and client-server technologies, companies expect their legacy systems to take advantage of these new technologies and cooperate with their heterogeneous systems and environments.

In this paper, we integrated a legacy tool in a heterogeneous distributed computing environment using wrapping. The wrapper around the tool was implemented in C using CORBA which provides a communication infrastructure that allows our tool to be accessed remotely. The legacy wrapping avoids rewriting the entire tool, but has modernized the tool to have the capability to cooperate with the customers' heterogeneous distributed computing environments.

We further reengineered our tool to enhance its functionality. The tool can be used to reengineer a legacy business application into a component-based application. Reusable rules are identified and the corresponding code is extracted out from the programs into CORBA components in COBOL which can be shared across different business applications for reuse.

Overall, we discussed how a centralized legacy tool was migrated into a heterogeneous distributed computing environment and how the tool was enhanced to support the business applications for the next generation computing.

References

[1] Chia-Chu Chiang, "Reengineering Enterprise Systems for Y2K Compliance," *The Chinese-American Academic and Professional Association in Southeastern States (CAPASUS '99)*, Invited Talk, Atlanta: Georgia, USA, July 9-11, 1999.

[2] Chia-Chu Chiang, "Encapsulating Legacy Systems for Use in Heterogeneous Computing Environments," *Information and Software Technology*, Vol. 43, No. 8, July 2001, pp. 497-507.

[3] Chia-Chu Chiang, "An Adapter for Implementing CORBA Objects in COBOL," *Proceedings of the Fifteenth International Conference on Systems Engineering (ICSEng 2002)*, August 6-8, 2002, Las Vegas: Nevada, USA, pp. 494-500.

[4] Chia-Chu Chiang, "The Use of Adapters to Support Interoperability of Components for Reusability," *Information and Software Technology*, Vol. 45, No. 3, March 2003, pp. 149-156.

[5] John Edwards, Paul Clements, Jack Gascoigne, and Ian Coutts, "Component-Based Systems – The Basis of Future Manufacturing Systems." *Proceedings of the Component User's Conference (CUC '96)*, July 15-19, 1996, Munich, Germany, pp. 105-110.

[6] Object Management Group, *http://www.omg.org,* 2002.

[7] X. Wang, J. Sun, X. Yang, Z. He, and S. Maddineni, "Human factors in extracting business rules from legacy systems". *Proceedings of 2004 IEEE International Conference on Systems, Man and Cybernetics* pp 200-205, 2004.

[8] M. Weiser, "Program slicing," *IEEE Transactions on Software Engineering*, Vol. 10, No. 4, pp. 352-357, July 1984.

I-Navigate: Intelligent, Self-adapting Navigation Maps

Herwig MAYR
Department of Software Engineering
Upper Austria University of Applied Sciences
Hauptstr. 117, A-4232 Hagenberg, Austria
E-mail: Herwig.Mayr@fh-hagenberg.at

Abstract

In the frame of our "I-Navigate" project, we create a software system that allows to extend digital navigation maps automatically and, thus, to improve their quality continuously. In order to achieve this, a track is gathered through tracing the paths of users (on foot, by car or by bicycle, for instance) via GPS. This track is converted into a route, which is merged with the current map graph and supplied to the user as an updated, navigable map.

Areas of application comprise the continuous extension of car navigation systems (last mile integration), the addition of off-road features to navigation maps (e.g. for outdoor and leisure activities, like mountain biking or off-track skiing), but also the surveillance of users navigating such a track in order to enable better targeted rescues in emergency cases (e.g. for mountaineers).

Keywords: GPS, Automobile and Pedestrian Navigation, Model-based Incremental Map Creation, ADAS

1. The Need for Incremental Creation and Adaptation of Navigation Maps

Automotive navigation systems, being developed since the late 1980s, are still among the key topics worked on in information technology. All common systems rely on a similar architecture: using a road database in the manufacturer's proprietary format, a route calculation module, a positioning module processing the Global Positioning System (GPS) data, and a map matching module (see [9]). Vehicle navigation systems can calculate a route along the roads in the database from the current car location to the destination point, and guide the driver via acoustic and visual instructions.

Even though navigation systems have become more reliable and affordable during the last few years, the navigation data market has remained rather inflexible, thereby slowing down the innovation process in the field of personalization of navigation systems and advanced driver assistance systems (ADAS), which are to become key aspects in the near future [2]. Thus, flexible, adaptive navigation using data that can be personalized individually (cf. Fig. 1), as well as extended for the public benefit still remains an interesting research topic with good chances for commercial value at a later stage.

2. Extending the Target Groups Using More General Models for Navigation Data

Computer-based navigation and (mobile) navigation devices are among the hottest topics currently worked on in information technology. One reason is the increased versatility and computing power of modern mobile devices (mobile phones, PDAs, etc.), the other being the considerable price reduction of these devices over the last few years making high-end mobile tools affordable to nearly everyone.

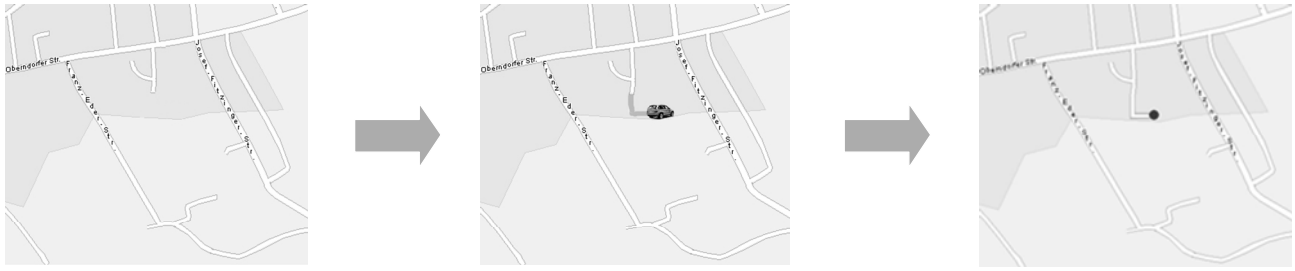


Figure 1: The need for individual navigation map extension

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Despite the rapid change of the market of navigation hardware, the availability of navigation software (i.e. route and map data) has remained rather constant over the last years: The suppliers of professional map data (mainly Navteq and Teleatlas) issue updates of their map data approximately every six months, mainly focusing on the needs of automobilists (Fig. 2). Individually recorded tracks are collected and distributed over the internet on a private, purely non-commercial basis (Fig. 3). These tracks can be analyzed and evaluated using suitable tools [10].

All navigation data are gathered storing satellite-defined global coordinates. The currently sole supplier of such satellite data is the Global Positioning System (GPS) by the U. S. Department of Defense (see [5] for details), the most popular reference coordinate system being the World Geodetic System 84 (WGS84), see [6]. Basically, two types of application have been developed utilizing the GPS signals:

1. **High-tech embedded systems**: Car navigation systems, for example, take the GPS signal as a basis for determining the car position on a predefined, stored map (Fig. 2). During driving, on-board sensor information is utilized for correcting the GPS deviation and, thus, increasing position accuracy considerably.

Figure 2: Professional navigation data for automobile use

2. **Low-tech mobile GPS devices:** Special-purpose GPS devices or PDAs/smartphones with built-in GPS receivers allow the storage and export of sequences of GPS coordinates (called *tracks*, Fig. 3). Such tracks can be used for point-to-point guidance of the user in order to retrace a stored track.

Both types of application are well developed and broadly used. However, up to now both worlds are in no way connected: Exchange of tracks or (sub-)maps is not possible.

However, the hardware gap between embedded navigation systems and mobile GPS systems gets increasingly smaller: The built-in navigation systems get better encapsulated and their interfaces more standardized (cf., e.g., the AUTOSAR initiative [1]), whereas mobile devices turn into fully capable, miniaturized computer systems with added sensor features like compass, accelerometer etc. Also, the need for joint systems increasingly arises: People want to use their mobile navigation systems in their car (utilizing better map data and sensor information), whereas people used to car navigation systems want to get navigation aid also outside their car.

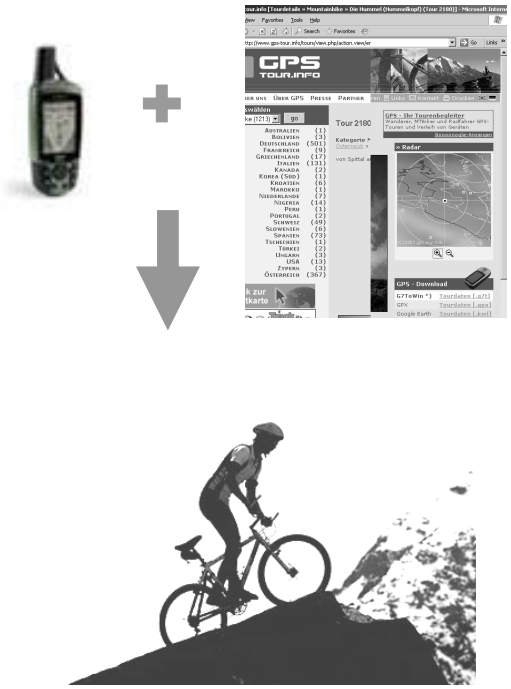


Figure 3: Non-commercial navigation data exchange for sports and leisure

The only current solution to update map data of (car) navigation systems is to supply update DVDs with completely or partially new map content (cf. a respective initiative of Navteq and DaimlerChrysler shown at ITS in San Francisco for a partial update). All these data are created offline; vehicle position data while moving are solely

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

used for determining whether a vehicle is "on-road" or "off-road", but not for determining possible road extensions.

Thus, our new approach utilizes recorded position data that is classified as "off-road" as a basis for extending navigation graphs. This necessitates the creation of navigation maps (and map extensions) from recorded tracks. The principal steps towards such a solution together with its algorithmic challenges will be discussed in the next section.

3. Solution Models and Methods

As mentioned in Section 2, basic GPS data consists of a sequence of so-called *GPS points*. Such a point describes a three-dimensional location anywhere on the earth. In addition to the three coordinates, typical GPS receivers also supply data of the current velocity of the object receiving the GPS data, the number of satellites used for computing the location information, and the estimated calculation error [5].

Each individual GPS point can be used as a target point for a GPS device. Thus, the device can use the deviation of the current position from the target position to indicate the direction and the distance of the target point from the current location. When a GPS device is used to record a sequence of GPS points (typically spaced one second from each other due to the GPS specification for non-military use), this sequence can be used for an iterative point-to-point navigation allowing the retracing of a stored track from one GPS point to the next.

In the following, we describe our three-step approach, how we utilize numerous recorded GPS tracks to incrementally create and extend navigation maps (see [7] for details):

Step 1: Route Conversion of a Single Track

Whereas a *GPS track* is comprised of a sequence of GPS points, we define a *GPS route* (short: route) to be a (linear) graph of vertices and edges, where the vertices are defined by the (modified) GPS points, and the edges connect two subsequent GPS points and may contain additional information (like terrain type, type of vehicle used for gathering etc.). Since for each GPS point, an error is specified by the GPS data, this error can be used for defining a circular environment around each point that describes possible modified locations.

Depending on the type of entity used for recording the route (pedestrian, bicyclist, etc.), a *mobility model* can be defined describing maximum velocity, probable and maximum curve radii depending on the velocity, among other parameters. According to this model, and additionally taking the reported velocity from the GPS data into account, the most probable GPS route within the limits of the GPS track data can be determined (cf. Fig. 4a), also containing a smoothed velocity profile of the entity.

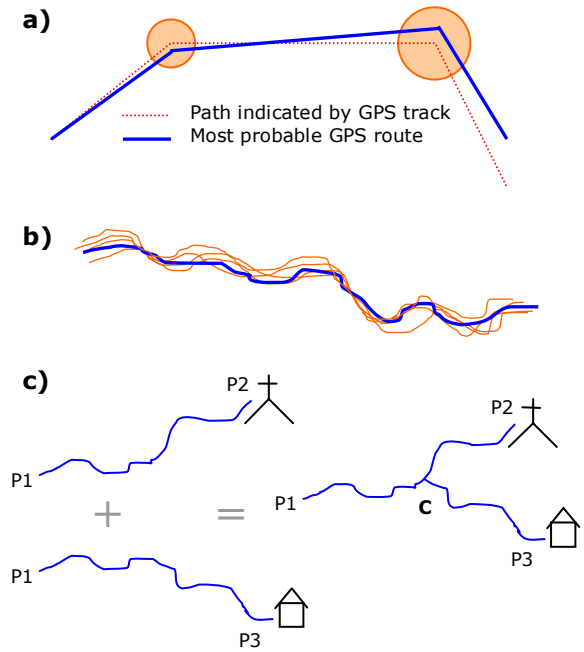


Figure 4: A three-step approach to individually created navigation maps

Step 2: Combining Routes from Multiply Recorded Tracks

If the same geographic path is recorded several times by the same entity or different ones, the most probable course of the real path or road can be determined by a suitable weighted averaging method (cf. Fig. 4b).

In order to enable this, additionally to the mobility models for each different entity contributing to the basic data, a *terrain model* has to be defined taking into account environment-specific factors that influence the GPS readings (like forests reducing the GPS quality, high-speed passages e.g. for cars) that add motion error to the GPS readings. If the behavior of the GPS receiver is reasonably well documented or investigated, a *receiver model* can be added to increase model adaptability and, thus, the correctness of the computed most probable course.

For creating and combining the models involved, we use the technique of *Kalman filtering* [4]. Kalman filtering allows the determination of an "optimal estimate" of a state from a number of observed measurements within a noise-induced system comprising system errors and measurement errors [4]. The main principle is to improve even the single best observed measurement taking into

account the (weighted) relevance of multiple measurements of the same circumstance. When using Kalman filtering for estimating the most probable path an object took while recording GPS information, the following models have to be established [4]:

- **System model**: The system model describes the possible sources of noise induced by the measuring device itself. In the frame of our GPS track analysis we call this model the *receiver model*.
- **Measurement model**: This model describes the noise that may be induced when applying the system in its environment. In the frame of our GPS track analysis we call this model the *terrain model*.
- **Target model**: This model allows the parameterization of the targeted entity and thus the calibration of the Kalman filtering algorithm to a certain dynamic process. Since in our application, GPS route data is created by mobile entities, we call this model the *mobility model*.

Whereas one generally needs only one receiver model (per type of measuring device) and one terrain model for a certain type of environment, a mobility model has to be defined for each class of mobile entities (e.g. pedestrian, bicycle, automobilist, rally driver). The mobility model adapts the estimation curve of the Kalman filter to the behavior of the entity (e.g. maximum velocity, curve radius, acceleration or deceleration in our application) and it constitutes the major part of setting up a suitable system [11].

The following aspects had to be particularly considered when adapting Kalman filtering models for our purpose:

- Depending on the type of entity, additionally to the maximum possible velocity a maximum distance limit between two GPS readings had to be set to compensate for unrecorded (missed) GPS positions. Such missing values can not be ignored when merging two (or more) tracks; instead; the most probable estimate has to be substituted.
- Two tracks are not matched according to their number or sequence of GPS points. Instead, an adapted *nearest-neighbor search* is applied. This not only compensates for different recording speeds (e.g. merging recordings from the same track first performed by a car, then by a bicycle), it also allows the inclusion of data from the same track recorded at the opposite direction!

Kalman filters are also used for determining joint segments of tracks and their respective crossroads. This will be detailed in the next section. For further (implementation) details on our Kalman filter implementation, we refer to [11].

Step 3: Combining Routes into a Graph Yielding a Navigation Map

Up to now, we have assumed that each recorded piece of track information denotes the same path or road in the real world (with the very same start point, end point, and direction traveled). However, routes may only partially overlap, be part of another, or be completely disjoint.

This problem is not considered at all in current navigation solutions: Current car navigation systems suppose the vehicle to be moving only on the roads contained in the stored map and reposition the vehicle onto the closest map segment, when the vehicle is considered off-road (so-called *map-matching*, cf. [9]).

Current portals for GPS tracks consider each stored or uploaded track (or route) individually. Thus, the user has to decide which route to take at the time of downloading. In the example of Fig. 4c, the decision whether to hike to the mountain top or to the hut has to be made at the beginning of the trek. No information is stored or can be derived that the first part of the two routes is identical. Thus, the decision, which way to go, cannot be delayed until reaching the crossroads **C**.

We base our incremental data update strategy on the EU-supported "ActMAP" project (see [8] for details). For map-matching, we utilize a sub-graph matching ("*registration*") algorithm that was originally developed for analyzing blood vessels of the human brain [3] and has been adapted for geographic data by us in the following way:

- Sections of the route have to be identified that comprise new elements with respect to the map. As a side effect, sections of the route that are already fully contained in the map are also identified and can be used to increase the quality of the map in the respective sections.
- The new elements have to be added ("registered") to the map, adapting and extending the graph appropriately (e.g. adding a new vertex and corresponding edges when the new route section bisects an edge of the already defined map graph). We map characteristic points of the new sub-graph into the existing routing graph using a distance mapping (chamfer) algorithm. For implementation details on this algorithm, we refer to [3].

In order to identify sections common to the map and the new route to be added, first a threshold must be defined when two GPS locations are to be treated as two definitely different locations in the real world. This can generally *not* be done fully automatically and needs human inspection (and sometimes correction). For this human intervention we devised a graphic visualization tool in order to better understand the topology of the individual case.

Such a threshold enables that when merging several tracks, which only partially cover each other, only the respective number of overlapping track segments are taken into consideration for the Kalman filtering. Also it handles the frequently occurring situation of a track to be merged being a sub-track of another (i.e. a partially walked hiking route already covered by the self-created map) without distorting the longer path. Additionally, this function indicates possible crossroads in the routes created, supplying a main functionality for defining the map graph.

4. Benefits from Our Approach

Our model-based approach allows the integration of various types of recorded data into one basic set and enables the computation of the most probable map graph representing real world paths and roads used by the recording entities. This results in the following major improvements:

- **Better Maps**: By individual additions of missing map segments as well as broadcasting them via a server (after verification), maps are being kept highly up-to-date, particularly for highly frequented new road segments.
- **Better Guidance**: Besides generalized optimization criteria (shortest route, scenic route, etc.), user-specific criteria can be used for route calculation allowing a highly automated personalization of the navigation system (e.g., routes considered safe by a driver).
- **Better Vehicle Efficiency**: Additional map information can be used to adapt the vehicle to the environment and conditions. Curve radii, for example can be used to adjust brake forces, or slope information may help adjusting the fuel injection system for optimal engine performance.
- **Intelligent Advanced Driver Assistance Systems (ADAS)**: Currently, ADAS rely mostly on other input sources, like sensors or cameras. A vehicle navigation system itself (being an ADAS) can also be a source of valuable information for other ADAS on board at the same time, when sharing the acquired data.

When not used "in the loop", i.e. for navigation and ADAS improvement solely for the car that collects the data (where the person collecting and using the data generally is the same and, thus can rather easily be made responsible for using the data at her/his own risk), solid validation of the data gathered is necessary.

Although plausibility checks can be performed on the data and data sets can be classified according to driver profession, vehicle type etc., manual inspection will never be able to be eliminated completely. Navteq Europe, for instance, is very interested in our collected data, because it allows them to prioritize the routes of their own data collection vehicles. In this way, Navteq map data will be updated in the same quality as up to now, but the roads chosen to be included will much better reflect customer needs.

Up to now, only a prototypic implementation exists for our data structure extensions and algorithms. However, this implementation is already integrated into a test bed of the current Siemens car navigation system and running in an actual car. Certain decisions and parameter adaptations within our algorithms must currently still be input manually, which necessitates a two-person operation at the current stage. Therefore it is too early for a quantitative assessment of our results.

Qualitatively, however, our system has already proven its ability to extend the navigation system's map data by roads driven that are not contained in the actual data (like the S1 circumnavigation freeway south of Vienna, Austria, that was opened to the public in spring 2006 and is not contained in any navigation map data available as of early 2007).

5. Applications

Fields of application of our approach include:

- the creation of personalized maps, e.g. for training courses for runners or cross-country skiers, route guidance for pedestrians with disabilities like wheel-chair users, visually-challenged or elderly people;
- harmonized maps from different sources (pedestrians, bicyclists, automobilists etc.); in the future even more accurate data from car navigation systems for ADAS;
- incremental update of the navigation model ("improving and extending the map with each usage").

One step towards an integrated solution is the in-the-loop usage of the data acquired by the same entity. In this way, a navigation system in a specific car can "learn" about the domains mentioned above and, thus, intelligently extend its navigation data base.

Our prototypes show that our approach constitutes a significant advantage with respect to route data collection for personal use as well as map portal servers and in the future may also allow continuously updated and personalized map information in car navigation systems.

The most important area of further improvement may be the utilization and collection of ADAS data, which will allow a considerable "personalization" of the car electronics, thus adapting the car behavior and performance to driver style, environment topography and conditions, for instance. According to the current project road maps at our industrial partners, first commercial versions of navigation systems that include our components shall be available early in the next decade.

6. References

[1] Convergence Transportation Electronics Association. *AUTomotive Open System ARchitecture – An Industry-wide Initiative to Manage the Complexity of Emerging Automotive E/E-Architectures*, CTEA white paper # 2004-21-0042, AUTOSAR Partnership, Munich 2004.

[2] eSafety Implementation Road Map Work Group. *Final Report and Recommendations of the Implementation Road Map Working Group*. Technical report, eSafety Forum Working Group for Directorate-General Information Society of the European Commission, October 2005.

[3] W. Friedl. *The Registration Process in Trans-cranial Doppler Zoography* (in German), Master's Thesis, Department of Medical Software Engineering, Upper Austria University of Applied Sciences, Hauptstr. 117, A-4232 Hagenberg, Austria, July 2004.

[4] M.S. Grewal and A.P. Andrews. *Kalman Filtering: Theory and Practice Using MatLab*, 2nd edition, John Wiley and Sons, 2001.

[5] B. Hofmann-Wellenhof, H. Lichtenegger, J. Collins. *GPS – Theory and Practice*. Springer Wien New York, 2001.

[6] International Civil Aviation Organization. *WGS84 Manual*, 2nd edition, ICAO Doc. 9674/AN 946, 2002.

[7] H. Mayr. *Model-Based Navigation Using GPS: One Step Closer to Intelligent, Incremental Maps*. In Proc. International Mediterranean Modelling Multiconference, pages 641-646, Oct. 4-6, 2006, Barcelona, Spain, 2006.

[8] H. Otto, L. Beuk, M. Aleksic, J. Meier, J. Löwenau, M. Flament, A. Guarise, A. Bracht, L. Capra, K. Bruns, and H. Sabel. *ActMAP ISO Input*. Technical Report D 3.3, ERTICO, November 2004.

[9] M. Quddus, R. Noland, W. Ochieng. *Validation of Map Matching Algorithms Using High Precision Positioning with GPS*, Journal of Navigation, vol. 58, pp. 257 - 272, 2005.

[10] P. Röbke-Dörr. *Navigation Using Smartphones and Java Phones* (in German), c't 17, pp. 170-179, 2005.

[11] K. Sonnleitner, M. Kofler. *Kalman Filtering for Merging GPS Tracks of Pedestrians (in German)*, Internal Report, Department of Software Engineering, Upper Austria University of Applied Sciences, Hauptstr. 117, A-4232 Hagenberg, Austria, June 2005.

Acknowledgements

Parts of this work have been financed by the Austrian government (FFG), grant no. 811 406 (FHplus programme). Other financially contributing partners to this project are Siemens Austria and Intersport Austria.

Know-how is also given by Siemens VDO, Regensburg, Germany, and Navteq Europe, Frankfurt. The author would like to particularly thank Charly Artmann of Siemens PSE Austria and Markus Schupfner of Siemens VDO Regensburg for all their initiatives and support of the project from the very beginning. Stefan Chiettini and Wolfgang Friedl of Siemens PSE Linz also lend a helping hand whenever needed (and this does happen frequently!).

Furthermore, a big "Thanks!" should go to the whole team working at this research project at the university research department in Hagenberg, namely Michael Kastner, Clemens Novak, Wolfgang Steinbichl, and Michal Vesely. Keep up the good spirit and the good work!

Model-Based Cyber Security

Galen Rasche, Erin Allwein, Michael Moore, Ben Abbott
Southwest Research Institute® (SwRI®)
grasche@swri.org

Abstract

This paper presents an approach for automatically verifying the correctness of cyber security applications through formal analysis guided by hierarchical models of the network, its applications, and potential attacks. This work is motivated by the need for a more intuitive, automated systems-level approach to determining the overall security characteristics of a large network. Given the complex nature of security tools and their general lack of interoperability, it is difficult for system designers to make definitive statements about the nature of their network defense. Our work focuses on creating an environment in which security experts can model the security aspects of complex networks using a graphical notation that is intuitive and natural for them, then automatically perform security activities such as formally verifying the safety of the network against known threats and exploring the network design for potential vulnerabilities. The environment is designed to utilize third party tools for performing these activities and concentrates on integration of these tools within a common modeling framework.

1. Introduction

Successful defense of large information systems requires a variety of tools and techniques. "Best of class" tools are offered by separate vendors to address specific network defense needs. These tools change rapidly in response to new network threats, with no one tool or approach offering sufficient overall security. For example, a host computer might run virus scanners, spyware detectors, and a host-based firewall, each developed by a different vendor, and with unique, specific (or at times overlapping and competing) goals. Numerous hosts usually exist within a large network and are commonly comprised of a variety of hardware, software, and operating systems with different vendor and vintage combinations. System-level protections, such as a network firewall and intrusion detection system, often exist as well.

The disjoint nature of this approach leads to difficulties in defending computer networks. The overarching problem is one of coordination of the many security applications and devices. The defense of large networked systems tends to involve a heterogeneous combination of tools that are not able to interoperate and cannot be controlled in a consistent manner. Further, system administrators often do not have a system-level view supporting reasoning about the system's capabilities of defense. In fact, combining tools together, while required for serious security, leads to such complexity that there is a trend in industry to outsource security to specialized organizations.

In this paper, we introduce an approach based on Model-Integrated Computing (MIC) that provides both host and system-level views of a network, as well the capability of automatically deploying and coordinating multiple network security tools. We describe a modeling framework in which a designer can assign security attributes to any level in a hierarchically described system and demonstrate how these models can be used to automatically drive existing theorem provers and reasoning engines to evaluate the level of protection against known threats, as well as potential vulnerabilities of a network.

2. Background

Formal methods, such as theorem proving and logic space analysis, show promise for formally verifying and analyzing the security characteristics of a system. However, the large scale of the information that is necessary to fully describe the networks, along with the complexity of the logical theorems that must be posed and maintained as formal logical theorems, causes this approach to be unrealistic if done by hand. This research automatically composes and executes the formal proofs necessary to determine the security

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

properties of the networked system. The proofs and the inputs to the formal proving tools to execute those proofs can be generated from models of the network, as long as the network models contain sufficient information.

One key aspect of this work is that in order to adequately and cost-effectively protect a large, complex network, many different types of activities should be undertaken; but these activities should be coordinated from a single knowledge base. For instance, one activity that is critical is to be able to determine if a network configuration is protected against known threats. This activity is performed fairly frequently as the network is reconfigured and as new threats are discovered. The accuracy and trustworthiness of the analysis results is critical to the protection of the network. This activity is not enough, however, as new threats are occurring continuously and with increasing frequency. Another type of activity is performing more speculative analysis to try and identify potential security breaches before the threats materialize. This type of analysis attempts to determine how a threat might break the current security measures based on the structure and attributes of the network and applications. Coordinating the analysis of performance against known threats and of potential weaknesses in defenses will create a powerful cyber security capability. This paper demonstrates how such a wall of security can be created using existing tools; namely, Prototype Verification System (PVS) for proving theorems about the security against known threats and Spin for analyzing potential weaknesses in defenses. We show how both of these activities can be coordinated from a model of the network.

Model-Based Development (MBD) techniques are used to create tools for modeling large, complex information systems using intuitive graphical notations. Visual modeling tools allow designers to easily describe and view the structure and attributes of assets contained within complex systems. Two competing philosophies in MBD exist. The first employs a common base modeling language for describing generic system concepts and allows for *domain profiles* (dialects) that are specific to a particular engineering discipline (e.g., network security). The predominant common modeling language is Unified Modeling Language (UML) [12]. This is the basis for Model-Driven Architecture, which uses UML notation. The benefit of using a common modeling language such as UML is that standard modeling tools can be used to model systems from many domains. One drawback is that UML is a software language and is not quickly understood by users from other domains. Secondly, any semantic that is applied to the UML models is necessarily independent of any particular domain. Any domain specific use requires the tools to be extended to support the semantic required by the domain (e.g., security analysis).

Several notable efforts have applied UML modeling to network security. In [6], UML extensions are used to evaluate systems for potential vulnerabilities. Stereotypes and tags are added to the UML system specification to create security requirements and model assumptions about the system environment. [7] provides a similar UML-based system for risk assessment. It builds from the CORAS project [11] to provide a framework in which the security of a system can be assessed from the security of its subcomponents. As was mentioned earlier, these modeling approaches are based on UML, which is not necessarily a language familiar to network security experts.

The second approach toward MBD is to design a modeling language specific for the application domain (e.g., network security) based on the commonly accepted notations and language already employed in that domain. Such a language is referred to as a Domain Specific Modeling Language (DSML). The benefit of using a DSML, as opposed to a UML-based notation, for modeling the security aspects of a computer network is that the modeling notation can be designed to be understandable, natural, and expressive to the individuals that are charged with doing the modeling and performing the day-to-day operations of protecting a network. Those individuals are likely not software experts trained in the syntax and semantics of UML. A side effect of employing a DSML is that modeling tools must be custom designed for the application. What makes this feasible is the existence of MIC technologies [5], which provide meta-programmable modeling tools. The Generic Modeling Environment (GME) [3], [13] is an MIC tool that is the basis for the cyber security modeling tool described in this work. MIC and GME are described in Section 2.1.

Our approach is to design a DSML for network security, leverage MIC technologies to build a Domain Specific Model Editor (DSME) based on that language, and create a framework for automatically analyzing the security of computer networks by deploying and coordinating multiple formal model-based analysis engines.

2.1. Model Integrated Computing (MIC)

MIC advocates the use of models in the construction, analysis, and synthesis of systems [1], [2], [3]. It uses domain-specific modeling languages

that are tailored to the needs of the specific application domain (e.g., computer security). Models are built by domain users (e.g., computer security operators) and are used in the analysis of designs and in the synthesis and integration of systems.

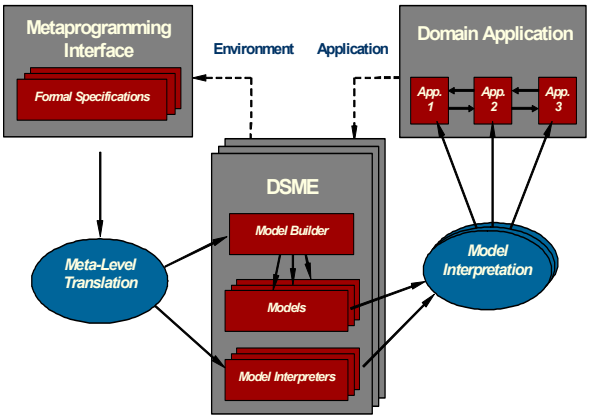


Figure 1. Generalized model-integrated computing approach

Figure 1 shows the typical flow of an MIC environment. The metaprogramming depicted provides formal specifications that are automatically synthesized into a domain-specific modeling environment (DSME). In the case of this research effort, GME is used to create a meta-model that defines a domain-specific modeling language for cyber security and to build the models. GME-supported tools are used to semi-automate the construction of model interpreters, which synthesize a logic-based version of the model.

2.2. Security analysis methods

Formal security analysis methods have long been used to verify portions of design-level specifications of systems [9]. They have also been applied to the verification of software and hardware systems. Of significant importance in the domain of cyber security is the capability to guarantee that the security applications that are deployed 1) meet the requirements as specified (i.e., fulfill the security needs of the design), 2) do not have adverse impact on the network or applications on the host system, and 3) are the optimal solution for the problem at hand given security applications available.

To perform this analysis, we have initially applied a formal theorem-proving technology utilizing the PVS [4]. We selected PVS because it has been widely used and is sufficiently rugged for use in large applications. PVS provides a specification language and theorem prover that allows for powerful automated deduction.

We recently began concurrent use of Spin [8], [10] for security policy verification. Spin has powerful state space analysis procedures that can be applied to automated analysis. It is also straightforward to create a network structure using reusable model definition components.

Our philosophy allows not only for "best of class" security tools, but also for verification methods.

3. System-level modeling of cyber security

The Cyber Security Modeling Tool was created by configuring GME with the Cyber Security Modeling Language (CSML). The CSML provides a framework in which to model the information assets and attributes of a large system. Figure 2 shows how MIC and CSML fit into the security modeling design flow. First, a model creator designs the model with the CSML. This is the step where the hierarchical network structure is created and the individual nodes are instantiated in the model. Next, a security tester specifies the test to be run based on the security policy and threat specification. The CSML interpreter uses the network model and the threat specification to generate code for a target application (e.g., PVS, Spin, etc.). The formal modeling tool uses the interpreter output as its input to run the security test. Based on the results, the security tester can revise security policies or models and repeat the verification process.

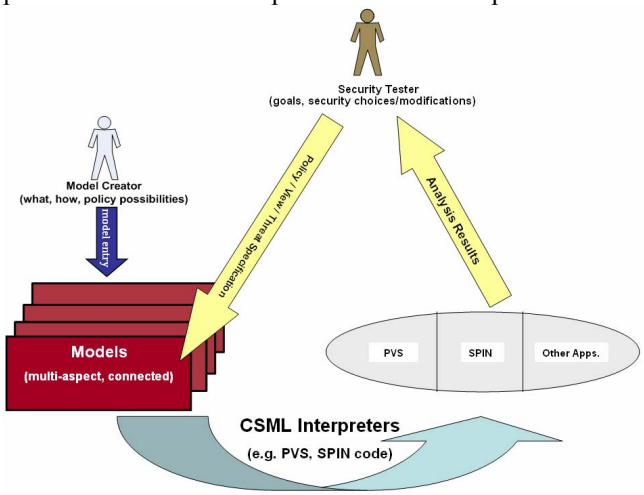


Figure 2. MIC approach to security modeling

3.1. Multi-Aspect Security Modeling

One distinct advantage of modeling with the GME tool is the ability to create different aspects. This allows the designer to control the presentation of

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

the network to the user, increasing the intuitiveness and usability of the model. While possible, it would be difficult to create a similar concept in UML-based models.

We have modeled four different aspects: physical, network shape, network isolation policy, and security suite. The physical aspect models the geographical characteristics, such as physical latency between networks. The network shape aspect focuses on the logical structure of the network. It is important to model both the physical and logical structure of the network to have a complete model of the system. While most work is focused on logical network shape, real users often want to know what the threat level is for a particular geographic region. Additionally, modeling "man-in-the-middle" portions of security threats requires reasoning about true physical location and access possibilities. This is particularly true for information systems that are distributed over large geographical areas. The network shape aspect is where the hardware and software characteristics of the system are modeled, associating applications with the nodes on the network. Within the network shape aspect, we model several general applications, such as mail servers, web servers, and file servers.

The network isolation policy aspect is used to model networks that are logically or physically separate. For example, a company may have a network that provides access to the Internet. It may also have a private network that is not supposed to be accessible from outside of the organization.

The security suite aspect models the security applications that are associated with each node. These would be programs such as virus scanners and firewalls. It would have been possible to group these with the other applications in the network shape aspect. However, it is more intuitive from a modeling standpoint to associate the security applications with each information asset in a different aspect.

3.2. Example network model

As an example, we will step through the different aspects of a simple network. The objects in the model are instantiations of network components, information assets, and applications that are modeled in the metamodel. These objects are essentially dragged and dropped into the DSME. When an object is created in the DSME, it is automatically put in the database of the tool. Since the different aspects of the model are connected, the object will appear there as well. Thus, the aspects are forced to be complete, helping to eliminate errors in the model. Portions of the icons can be specified as "parameters" (e.g., IP address). These parameters are attributes in the metamodel that are added to icons and connectors. The model that is stored in the tool's database can be output in XML format, allowing a variety of tools to parse the model.

Figure 3 shows the top-level view of our network. We model two separate offices for a sample company that are connected through the public Internet infrastructure. We have also included a botnet on the external network for modeling attacks on the network.

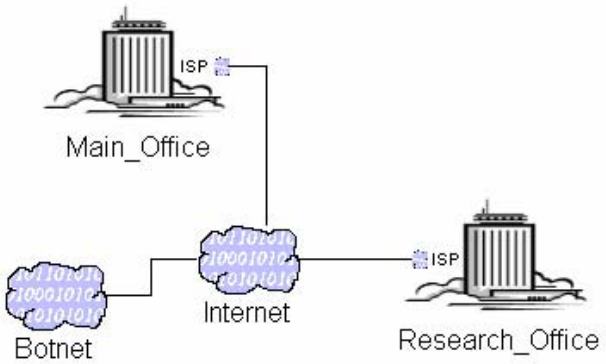


Figure 3. Top-level view of model

The focus of this example will be the research office. Figure 4 shows the network shape aspect of a simple network in the research office. For this example, we only include three nodes protected by a network firewall.

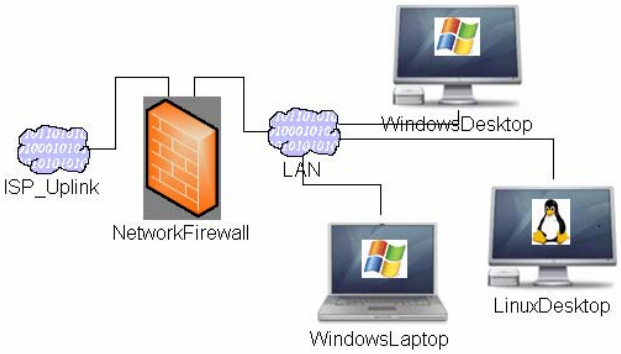


Figure 4. Network shape aspect of Research_Office

The attributes of the information assets in the model can be set. For example, identifying characteristics of the three nodes, such as operating system, IP address, and MAC address, can be specified. Figure 5 shows a sample set of attributes for the network firewall.

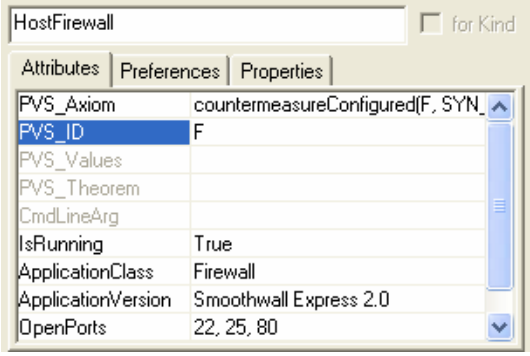


Figure 5. Example firewall attributes

Generic applications can be associated with each information asset in the network. Figure 6 shows a few sample applications that are included in the model. Attributes can be set for the applications, such as the vendor, application name, and application version.

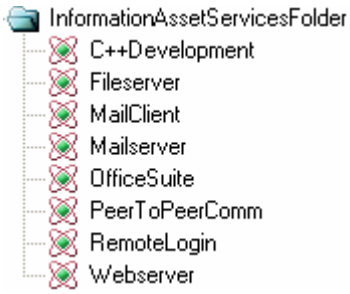


Figure 6. Sample applications for an information asset

The security suite aspect of the model, as shown in Figure 7, demonstrates how the individual information assets are associated with different sets of security applications.

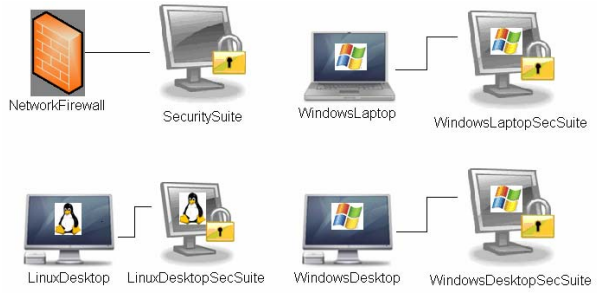


Figure 7. Security suite aspect

The various security suites can be given default security applications based on their platform and operating system. Figure 8 shows a sample of the security applications. Instances of the security applications can be dropped onto each host, including the network firewall.

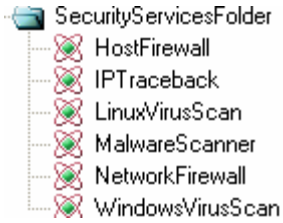


Figure 8. Security applications

In summary, a multiple aspect model describing the network is created. The model is actually a database that can be processed and reasoned about from a variety of directions. The complexity of generating this model from a single aspect in a classic text-based declarative fashion would be error prone and difficult for the typical front-line IT professional.

4. Model-driven security analysis

We are automating the process of analyzing the security of large systems. Such systems consist of so many individual devices, interconnections, and applications that one cannot realistically generate a logical description of the system manually. A typical system will require hundreds of individual proofs to determine that all of the security goals are met. To manage such proofs by hand would be intractable. Thus, automation is used to generate the description of the system and to execute the proofs of the security requirements.

An interpreter is used to transform the GME model into a partial specification of the system defined in the language of a theorem prover such as PVS. It defines types for the various elements of the model, instances of those types, connections between the elements, and properties of the elements (as logical predicates).

For PVS, the rest of the specification is created by hand. The manually created part of the specification defines the rules that govern security interactions between components of the system. The rules are used to reason about a system's protections and vulnerabilities. For example, they can define that a particular countermeasure can block a particular network attack, that a server or firewall can implement countermeasures, and that a firewall protects a network. The requirements that must be met are defined as theorems which can be tested using the theorem prover.

Requirement goals for a subject system, expressed as theorems, have varying levels of detail. Goals can

be very general, such as requiring that protected information flows are not vulnerable, that failures in one part of the system do not propagate throughout the system, or that critical systems maintain at least a minimal level of functionality while under attack. Very specific questions can also be explored using the GME model, such as determining whether a particular server will be taken out by a denial of service attack.

4.1. Example automated PVS proof

As an example of exploring reasoning over the security properties of a system with a firewall, such as the one described in Section 3.2, assume a subject system with a network N comprised of servers $S_1 \ldots S_m$. N is protected by firewall F. If one of the servers, S_1, is the target of a network attack for which there is a known countermeasure, we want the system specification to show that the attack is not successful.

We define the predicates *member(Server, Network)* and *safe(Server, Attack)* to define the components that make up a network and to indicate whether or not a particular server is safe from a specified attack. A network can also be safe from an attack: *safe(Network, Attack*), and it can have a firewall: *hasFirewall(Network, Firewall)*. A firewall's configuration is defined by the configuration of its countermeasures: countermeasureConfigured*(Firewall, Countermeasure),* and its ability to counter a given attack is given as *blocks(Firewall, Attack)*. The relationship between an attack and a successful countermeasure against it is given as *countermeasure(Countermeasure, Attack)*. The following example uses a SYN flood as the attack and a SYN proxy as the countermeasure. The axioms describing the security relationships of elements in the system are given in ax1-ax4. These are the types of rules that would be defined by hand. The axioms that instantiate the deployment of the systems are given in ax5-ax7. These are automatically generated by the interpreter. The PVS specification is given below.

```
syn_flood: THEORY
BEGIN
% Object Types
Network, Server, Firewall, Attack,
Countermeasure: TYPE

% Constants
SYN_FLOOD: Attack; SYN_PROXY:
Countermeasure

n: VAR Network; s: VAR Server
f: VAR Firewall; a: VAR Attack;
c: VAR Countermeasure
N: Network; S1: Server; F: Firewall

member: [Server, Network -> bool]
safe:   [Server, Attack -> bool]
safe:   [Network, Attack -> bool]
hasFirewall: [Network, Firewall -> bool]
countermeasureConfigured: [Firewall,
Countermeasure -> bool]
blocks: [Firewall, Attack -> bool]
countermeasure: [Countermeasure, Attack
-> bool]

ax1: AXIOM FORALL f,c,a:
countermeasure(c,a)
     & countermeasureConfigured(f,c)
IMPLIES
     blocks(f,a)
ax2: AXIOM FORALL f,a,n:
hasFirewall(n,f) &
     blocks(f,a) IMPLIES safe(n,a)
ax3: AXIOM FORALL s,n,a: member(s,n)&
     safe(n,a) IMPLIES safe(s,a)
ax4: AXIOM countermeasure(SYN_PROXY,
     SYN_FLOOD)

ax5: AXIOM countermeasureConfigured(F,
     SYN_PROXY)
ax6: AXIOM hasFirewall(N, F)
ax7: AXIOM member(S1, N)

% Conjecture: S1 is not vulnerable to
SYN flood attack
thm1: THEOREM safe(S1, SYN_FLOOD)

END syn_flood
```

The theorem *safe(S_1, SYN_FLOOD)* can be proven in PVS by instantiating the universally quantified axioms and simplifying. An example proof is given below.

```
;;;    Proof    thm1-1    for    formula
syn_flood.thm1
;;;  developed  with  shostak  decision
procedures
("
 (lemma ax7)
 (lemma ax6)
 (lemma ax5)
 (lemma ax4)
 (lemma ax3)
 (lemma ax2)
 (lemma ax1)
 (inst -1 "F" "SYN_PROXY" "SYN_FLOOD")
 (inst -2 "F" "SYN_FLOOD" "N")
 (inst -3 "S1" "N" "SYN_FLOOD")
 (bddsimp))
```

This is a simplistic example that is easily generated by hand. A realistic example might involve hundreds of networked servers and hosts and many security requirements. Fully automated reasoning is necessary to make such a problem tractable. This requires proof tactics which can handle the multitude

of axioms generated from such a system and that can reason about aggregations of components and their security properties.

4.2. Example using spin

In addition to the theorem prover PVS, we have been integrating Spin into our MIC environment. Spin is specifically designed for the verification of distributed systems, making it a natural choice for modeling networks. Spin also has an easy to use command-line interface, making system verification much simpler to automate.

This simple example focuses on modeling the protective behavior of the firewall in the model in Section 3.2. An attack on ports 21 and 25 coming from the botnet (Figure 3) has been modeled. For identification, the attacks on ports 21 and 25 are tagged with the numbers 245 and 246, respectively. The interpreter is used to generate a Spin input file from the model. The attributes of the botnet are used to indicate the target machines and port numbers. The local area network (LAN) in Figure 4 is parsed to determine what machines are connected to the firewall. Additionally, the attributes of the firewall, shown in Figure 5, declare which port numbers will pass traffic. The code generated for the firewall is shown below.

```
active proctype Firewall() {
  short port, msg;
end:do
  ::enet?WindowsDesktop,port,msg ->
     if
       ::(port == 22) -> inet1!port,msg;
       ::(port == 25) -> inet1!port,msg;
       ::(port == 80) -> inet1!port,msg;
       ::else     ->     printf("Firewall
blocked %d on WindowsDesktop on port
%d\n",msg,port);
     fi;
  ::enet?WindowsLaptop,port,msg ->
     if
       ::(port == 22) -> inet2!port,msg;
       ::(port == 25) -> inet2!port,msg;
       ::(port == 80) -> inet2!port,msg;
       ::else     ->     printf("Firewall
blocked %d on WindowsLaptop on port
%d\n",msg,port);
     fi;
  ::enet?LinuxDesktop,port,msg ->
     if
       ::(port == 22) -> inet3!port,msg;
       ::(port == 25) -> inet3!port,msg;
       ::(port == 80) -> inet3!port,msg;
       ::else     ->     printf("Firewall
blocked %d on LinuxDesktop on port
%d\n",msg,port);
     fi;
  od;
}
```

Once the interpreter has finished generating the code, Spin can be automatically run from the MIC environment. The following output is produced for the simple attack modeled in Figure 3 and Figure 4:

```
/* running spin simple_attack.pr */
  Firewall blocked 245 on WindowsDesktop
on port 21
  Firewall blocked 245 on WindowsLaptop
on port 21
  Firewall blocked 245 on LinuxDesktop
on port 21
      attack2 attempted on port 25:
              Server dying
      attack2 attempted on port 25:
          Server dying
      attack2 attempted on port 25:
                  Server dying
  timeout
#processes: 1
                queue 1 (enet):
                queue 2 (inet1):
                queue 3 (inet2):
                queue 4 (inet3):
 54:      proc   0 (Firewall) line   31
"simple_attack.pr" (state 34) <valid
end state>

5 processes created
```

This example is primarily meant to demonstrate how code for formal verification can be generated from a MIC model. The complexity of the model can be easily increased by adding more attributes to the nodes and application models. The additional SPIN code would be automatically generated, thereby avoiding user errors and reducing complexity.

5. Conclusion

The paper has described our current work in utilizing a model integrated computing approach to describe what we know about our system architecture and its vulnerabilities in a hierarchical, intuitive way. The model is used to drive third party verification tools to test for known threats and to identify unknown weaknesses. This is our first step along the path. Subsequent work will not only refine the formalism and verification, but also will include mechanisms to automatically deploy, modify, and redeploy the current security tools and mechanisms.

References

[1] G. Karsai, F. DeCaria, "Model-Integrated Online Problem-Solving Environment for Chemical Engineering," *IFAC Control Engineering Practice*, Vol. 5, No. 5, pp. 1-9, 1997.

[2] G. Karsai, J. Sztipanovits, S. Padalkar, C. Biegl, "Model Based Intelligent Process Control for Co-generator Plants," *Journal of Parallel and Distributed Systems*, pp. 90-103, 1992.

[3] A. Ledeczi, M. Maroti, G. Karsai, G. Nordstrom, "Metaprogrammable Toolkit for Model-Integrated Computing," Proceedings of the Engineering of Computer Based Systems (ECBS) Conference, pp. 311-317, Nashville, TN, March, 1999.

[4] S. Owre, J. M. Rushby, and N. Shankar, "PVS: A prototype verification system," In Deepak Kapur, editor, *11th International Conference on Automated Deduction (CADE)*, Vol. 607 of *Lecture Notes in Artificial Intelligence*, pp. 748-752, Saratoga, NY, June 1992, Springer-Verlag.

[5] J. Sztipanovits, G. Karsai, "Model-Integrated Computing," *IEEE Computer*, pp. 110-112, April, 1997.

[6] J. Jürjens, "UMLsec: Extending UML for Secure Systems Development," UML 2002, Dresden, Sept. 30 – Oct. 4, 2002, LNCS.

[7] M. S. Lund, F. den Braber, K. Stølen, "A Component-Oriented Approach to Security Risk Assessment," In Proc. 1st International Workshop on QoS in CBSE 2003 (QoSCBSE'2003), organized in conjunction with Ada-Europe 2003, pp. 99-110, Cépadues-éditions, 2003.

[8] G. J. Holzmann, "The Model Checker SPIN," IEEE Transactions on Software Engineering, Vol. 23, No. 5, pp. 279-295, 1997.

[9] C. Meadows, "The NRL Protocol Analyzer: An Overview," Journal of Logic Programming, Vol. 26(2), pp. 113-131, 1996.

[10] G. J. Holzmann, "The SPIN Model Checker: Primer and Reference Manual," Addison Wesley, 2003.

[11] R. Fredriksen, M. Kristiansen, B. A. Gran, K. Stølen, T. A. Opperud, T. Dimitrakos, "The CORAS Framework for a Model-Based Risk Management Process," In Proc. Computer Safety, Reliability and Security (Safecomp 2002), LNCS 2434, pp. 94-105, Springer, 2002.

[12] UML information and documentation on the web at http://www.uml.org/

[13] GME information and documentation on the web at http://www.isis.vanderbilt.edu/Projects/gme/

Secure Communication Trees in Ad Hoc Networks

Jan NIKODEM, Maciej NIKODEM
{jan.nikodem, maciej.nikodem}@pwr.wroc.pl
Wrocław University of Technology
The Institute of Computer Engineering, Control and Robotics,
Wybrzeże Wyspiańskiego 27, 50-370 Wrocław, Poland

Abstract

In this paper we examine the communication trees widely used in ad hoc networks. These trees can be used in a number of ways e.g. to route or broadcast messages and to aggregate information. In this paper we propose a new key distribution protocol for large and dynamic ad hoc networks over unreliable channel. Protocol proposed here involves broadcast exclusion based on polynomial interpolation and secret key cryptography.

Proposed scheme ensures security of broadcast and one-to-one communication, message authentication and gives a possibility to verify routing information. It also ensures t-resilience which means that up to t nodes can be compromised without influencing the security.

1. Introduction

Advances in ad hoc networks technology have enabled the mass production of small devices equipped with sensing and communication devices. Such devices can be deployed and form a spontaneous network capable of performing distributed sensing tasks. Because of those capabilities ad hoc networks are perfect candidates for operating in environments without any network infrastructure or where access is difficult. On the other hand many applications requires that data must be exchanged between nodes in a secure manner due to an adversary who may intercept communication or inject false messages. Therefore security becomes one of the most important concerns. However establishing secure communication is a challenging task since nodes have constrained power and computational capabilities.

A common method to ensure security in the ad hoc networks is to encrypt and authenticate all communication. This can be done in many ways since different encryption algorithms and authentication schemes have been proposed so far.

Usual solution that ensures authentication is based on message authentication codes (MACs). This are keyed hash functions that ensure receivers that transmitter knows the correct encryption key. The advantage of such solution is simplicity and low computation complexity while the drawback is a common key that has to be shared among the nodes.

Secure communication is usually achieved due to secret key cryptography and also requires that nodes share common encryption key. Moreover pairwise encryption keys are not enough since one-to-one communication is time and power consuming. Therefore there are also group keys utilized to encrypt broadcast communication. Group keys reduce the communication overhead per one message but on the other hand involve additional procedures to establish or change the key – so called group rekeying schemes.

The challenge is to distribute keys to all nodes as well as the network member's management (i.e. how to add and exclude nodes). The later is even more difficult but very often this two problems are treat as one.

2. Related Work

All existing group rekeying and network management schemes are based on one of five approaches: secret key approach, certificate based approach, Generalized Diffie-Hellman (GDH) based approach, broadcast exclusion approach and probabilistic approach. Unfortunately all of them have drawbacks.

Secret key approaches that utilize only one key allow to perform rekeying and node addition only. Since all nodes share the same key, there is no secure way a base station (BS) can distribute new key to some subset of all nodes. Logical tree hierarchy (LTH) is a variant of secret key approach in which nodes are organized in the tree structure. This allows node exclusion but requires large communication and storage overhead. Moreover LTH is not suitable for mobile ad hoc networks (MANETs).

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

New secret key approach was recently presented by Dimitriou [8] who proposed an algorithm for establishing communication trees that ensure security and authentication. His proposal allows to change network structure, node addition and exclusion. Disadvantage of this scheme is that it requires a common master key to be shared between nodes during initialization. Therefore protocol is secure as long as no node will be compromised during this process.

Public key approaches (certificate and GDH based) have large computational complexity and are not efficient in case of large networks. Large communication overhead on node exclusion is another disadvantage. Current research aim to decrease this overhead by using more efficient variants of GDH algorithms [2].

Broadcast exclusion protocols [5, 7, 13, 20] typically assume that the network structure is known. However in case of the ad hoc networks this requirement is very difficult to fulfil especially when nodes are deployed in a random manner (e.g. dropped from an aircraft). Another disadvantage of the broadcast exclusion protocols is that they were proposed to solve problems in one-to-many communication (e.g. pay-TV) where only one broadcaster exist. In case of the ad hoc network one central node is likely to become a bottleneck of the whole network. The advantage of broadcast communication is low communication and storage overhead. Computational complexity is usually larger then in case of other approaches but it can be reduced using different techniques (see e.g. [20]).

Recently Curtmola and Kamara [7] proposed to implement broadcast exclusion for ad hoc networks based on erasures codes. This ensures low computation complexity but limits number of successive transmissions that can be missed without loosing the opportunity to recover the key.

Last approach is based on a large set of symmetric keys like GKMPAN [23]. Each node in the network stores random subset of these keys. Assuming that each node obtains keys independently at random, knowing the number of keys in the set and number of nodes we can estimate the probability that after deployment neighbour nodes will share at last one common key. This happens with high probability (usually about 95–99%) allowing neighbour nodes to communicate directly and securely. If direct communication is impossible then nodes establish a multi-hop communication path using nodes that share at last one common key. In those schemes both storage and communication complexity is higher then in case of the broadcast exclusion. On the other hand the advantage of the probabilistic approach is low computation complexity.

In this paper we present a novel group rekeying scheme for ad hoc networks based on secret key and broadcast exclusion approaches. Our approach utilize secret keys to ensure authentication and privacy in one-to-one and one-to-many communication while broadcast exclusion, based on polynomial interpolation, ensures secure way for network management and group key exchange. Similar to [7, 8, 23] nodes that miss rekeying messages (e.g. due to lossy links or network partitions) or join the network are able to get group key through local communication. Immunity to coalitions of up to t malicious nodes (t-resilience) is an additional feature of our protocol.

3. Network and Security Assumptions

3.1. Network Assumptions

In this work we assume that the ad hoc network is spread over large area that exceeds communication range of a single node. Communication between nodes is thus performed in a multi-hop way with messages sent from one node to another.

Another assumption concerns central node or computer that plays crucial role in many rekeying protocols – usually called key server or group manager [16, 23]. Its main task is to coordinate cryptographic protocols and algorithms such as node addition, exclusion or rekeying. That is why the group manager is vulnerable to attacks thus likely to become a bottleneck of the whole network. To get rid of this threat we assume that there is no dedicated single group manager in the network and its tasks can be performed by any node in the network.

Finally we assume that nodes have limited power, storage, communication and computational capabilities. Thus nodes cannot afford complex operations and do not have enough space to store all pairwise secret keys. However we assume that every node has possibility to store a few kilobytes of secret data used to ensure security and authenticity.

3.2. Security Assumptions

We assume that every node can perform a role of a group manager managing the network membership and the group key used to secure the broadcast communication. In our protocol new nodes can join the network through communication with any node that already belongs to the network. On the other hand exclusion and session key rekeying is initiated by one or more nodes. We do not specify the cause for these operations but we propose methods how malicious nodes can be detected.

We also assume that the attacker can obtain all the secret information stored in the compromised nodes, and also that the non-compromised nodes are trusted (i.e. they execute protocol correctly). Finally we assume that all the compromised nodes are detected immediately by at least one non-compromised node. As we are going to present our proposal is immune to an adversary which takes control over up to t nodes, where t is a security parameter chosen

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

during the system initialization. Since we are interested in wireless ad hoc networks we also assume that an adversary can eavesdrop all communication.

4. The Protocol

In this section we present our protocol for building secure communication trees in the ad hoc network. The protocol consist of three phases: the preprocessing during which the base station determines system parameters and secret data for each node. The initialization phase when nodes determine pairwise keys for communication with each other and the BS. Third phase when nodes use these keys to perform a network organization and establish routing paths.

4.1. Preprocessing

This phase is performed prior to the network deployment. During this stage random prime integer p and generator α of the Z_p group is chosen. Afterwards every node u obtains p, α and two distinct polynomials – $g_u(x)$ and $h_u(y)$. These polynomials are product of an surface $z = f(x, y)$ intersection with planes $y = u$ and $x = u$ respectively,

$$\begin{aligned} g_u(x) &= f(x,y)|_{y=u} \\ h_u(y) &= f(x,y)|_{x=u}. \end{aligned} \tag{1}$$

For simplicity we assume that degrees of those polynomials are equal for every node in the network. We denote $\deg(g_u(x)) = \deg(h_u(x)) = t$. Polynomials (1) are stored in node's memory and afterwards the network is deployed.

In this way any two nodes u_1 and u_2 get pair of polynomials $\langle g_{u_1}(x), h_{u_1}(y)\rangle$ and $\langle g_{u_2}(x), h_{u_2}(y)\rangle$ that intersect in exactly two points:

$$\begin{aligned} g_{u_1}(u_2) &= f(u_2, u_1) = h_{u_2}(u_1) \\ h_{u_1}(u_2) &= f(u_1, u_2) = g_{u_2}(u_1). \end{aligned} \tag{2}$$

In this way each pair of nodes posses two pairwise secret keys.

4.2. Network Initialization

Network initialization enables nodes to determine pairwise secret keys used in one-to-one communication.

To determine key used to secure communication between node u_i and the BS, u_i has to compute:

$$\begin{aligned} K_{BS,i} &= g_{u_i}(0) \\ K_{i,BS} &= h_{u_i}(0). \end{aligned} \tag{3}$$

The same key can be determined by the BS by computing:

$$\begin{aligned} K_{BS,i} &= f(0, u_i) \\ K_{i,BS} &= f(u_i, 0). \end{aligned} \tag{4}$$

Pairwise keys between neighbour nodes are determined using a hello message. This is a broadcast message consisting only of node's id u_i and random $nonce$ to ensure immunity against replay attack:

$$Node_{u_i} \rightarrow Neighbours : \langle u_i, nonce\rangle .$$

Upon receiving this message each neighbour node u_j computes common secret keys $K_{i,j} = g_{u_j}(u_i)$ and $K_{j,i} = h_{u_j}(u_i)$ and response with the acknowledgement message:

$$Node_{u_j} \rightarrow Node_{u_i} : \left\langle u_j, E_{K_{i,j}}\left(u_j, nonce\right)\right\rangle .$$

When receiving this message node u_i can verify the identity of u_j by computing the encryption keys $K_{i,j}$ and $K_{j,i}$ and decrypting the message. In this way it is ensured that u_j belongs to the same network. The last step is a response to u_j to prove u_i's identity:

$$Node_{u_i} \rightarrow Node_{u_j} : \left\langle u_i, E_{K_{j,i}}\left(nonce + 1\right)\right\rangle .$$

As show in this example both nodes use two different keys $K_{i,j}$ and $K_{j,i}$ to prove their identity. Random $noce$ used in above protocol is used to ensure security against replay attack.

The last step of initialization process is to establish a group keys between the BS, nodes u_i and all of their neighbour nodes u_j. This can be done in a very simple way – sending a separate messages to each neighbour encrypted with one of secret keys $K_{i,j}$ or $K_{j,i}$. Unfortunately this is a naive solution that waste nodes power and increase communication overhead since the same information has to be transmitted unnecessarily many times.

A much more effective way for group key establishment can be achieved if we take an advantage of fact that all secret keys $K_{i,j}$ for every neighbour u_j belong to the same polynomial of degree t.

We propose to exchange group key K_i^G according to the following broadcast encryption scheme:

- node u_i selects a random integer $r < p - 1$,
- node u_i determines the set of pairs $\Phi = \{(x, y)\}$ of cardinality $\|\Phi\| = t$ as follows:
  - determines the set of excluded neighbour nodes $\Theta = \{u_j\}$ such that $\|\Theta\| \leq t$,
  - for each neighbour node $u_j \in \Theta$ adds to Φ a pair: $\left\langle u_j, \alpha^{r g_{u_i}(u_j)} \bmod p\right\rangle$,
  - adds $t - \|\Theta\|$ pairs $\left\langle x, \alpha^{r g_{u_i}(x)}\right\rangle$ to Φ for a random x such that $x \neq u_i$ for any node u_i.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

- selects a group key $K_i^G < p$ and encrypts it as $K_i^G \cdot \alpha^{rg_{u_i}(0)} \bmod p$
- broadcast a message:

$$Node_{u_i} \rightarrow Neighbours : \left\langle u_i, \alpha^r, \Phi, K_i^G \alpha^{rg_{u_i}(0)} \bmod p \right\rangle.$$

Upon receiving the message each neighbour node u_j can reconstruct the group key using Lagrangian interpolation in the exponent. The decoding procedure goes as follows:

- each node u_j uses its own secret data to compute a pair $\left\langle u_j, \alpha^{rh_{u_j}(u_i)} \bmod p \right\rangle$ and adds it to the set Φ (note that $h_{u_j}(u_i) = g_{u_i}(u_j)$),
- uses $t+1$ pairs (x, y) from Φ to reconstruct the group key as follows:

$$K_i^G = \frac{K_i^G \cdot \alpha^{rg_{u_i}(0)}}{\prod\limits_{k \in \Phi} \alpha^{rg_{u_i}(u_k) \prod\limits_{l \in \Phi, l \neq k} \frac{0-u_l}{u_k-u_l}}} \bmod p. \quad (5)$$

Above procedure allows to decode the correct group key only if node u_j was not among excluded nodes ($u_j \notin \Theta$). On the other hand, if node u_j is excluded then his pair $\left\langle u_j, \alpha^{rh_{u_j}(u_i)} \bmod p \right\rangle$ is in the set Φ broadcasted. Therefore u_j posses only t different pairs instead of $t+1$ required for reconstruction.

Above procedure requires only one broadcast communication with message size $O(t)$ to distribute group key to all neighbours. Moreover the same procedure can be used subsequently to change the group key or to exclude some nodes from future communication. Proposed scheme allows to exclude up to t nodes through distributing new group key to everyone except selected nodes. Broadcast exclusion scheme proposed here has a feature that identifiers of excluded nodes are broadcasted in a plain text. This is a drawback in standard applications of broadcast exclusions because of lack of anonymity. On the other hand in ad hoc networks this is an advantage since all nodes will know immediately which nodes are excluded.

At this point key establishment phase is complete. Nodes share information about neighbour nodes, determined pairwise secret keys and group keys. The next step is to build communication tree in order to create paths for communication between nodes and the BS.

4.3. Building a Communication Tree

Ad hoc networks are usually bigger then communication range of a single node. Therefore it is necessary to build a multi-hop structure that allows to communicate between distant nodes. In this paper we focus our attention on tree-like structure for wich cost effective routing algorithm have been proposed [6, 21, 22]. The protocol for creating a communication tree is a straightforward extension of the one presented in [6] and [22].

4.3.1. Clustering

Clustering divides the whole network into groups (clusters) of nodes. Each cluster consist of exactly one cluster head (CH) and few regular nodes (RN). Since CHs are responsible for inter and intra-cluster communication thus their selection has to be based on the energy level. This is a difficult due to fact that nodes are left unattended and have no information about energy of other nodes. Clever protocol (called HEED) to solve this problem was presented by Younis and Fahmy [22]. Their protocol selects CHs based on residual energy, terminates within fixed number of iterations and guarantees good cluster distribution. HEED protocol favourites nodes with highest energy but on the other hand it does not guarantee that the strongest nodes will become a CH.

The whole protocol goes as follows:

- each node u_i calculates the probability of becoming a CH:

$$P\left(u_i \text{ is CH}\right) = \max\left(C_{prob} \cdot \frac{E_{u_i}}{E_{max}}, p_{min}\right)$$

  where $C_{prob} = 0.05$, p_{min} is initially set to 10^{-3}, E_{u_i} is actual node's u_i energy and E_{max} is maximum node's energy (see [22] for details on those parameters),
- repeat following steps N_{iter} times:
  - each node u_i selects a number $a_i \in (0, 1)$ independently at random,
  - if $a_i < P\left(u_i \text{ is CH}\right)$ then node u_i becomes a CH and broadcasts a message:

$$Node_{u_i} \rightarrow Neighbours : \left\langle u_i, E_{K_i^G}\left(u_i, nonce\right) \right\rangle.$$

    Otherwise node listens to the broadcast messages and joins the cluster head heard,
  - if node become neither CH nor RN then it doubles its probability of becoming a CH ($P\left(u_i \text{ is CH}\right)$) and enters the next iteration of the protocol,

As it was presented by Younis and Fahmy above procedure terminates in fixed number of rounds $N_{iter} < \left\lceil \log_2 \frac{1}{p_{min}} \right\rceil + 1$. After this procedure every node is either a regular node or a cluster head.

Comparing to the standard HEED protocol our extension authenticates messages broadcasted by CHs. This is possible since each RN can verify whether u_i is encrypted in the message.

4.3.2. Building the Communication Tree

After network is divided into clusters BS constructs a broadcast message:

$$BS \rightarrow Neighbours : \left\langle BS, E_{K_{BS}^G}\left(BS, dist_{BS}, cost_{BS}\right)\right\rangle$$

where $dist_{BS}$ is a distance to the BS and $cost_{BS}$ is an overall energy cost of communication with BS [6]. Every node that receives this message decrypts it with a broadcast key and sets its $dist_i$ to $dist_{BS} + 1$ and $cost_i = cost_{BS} + cost(u_i, BS)$. Afterwards it broadcast new message:

$$Node_{u_i} \rightarrow Neighbours : \left\langle u_i, E_{K_i^G}\left(u_i, dist_i, cost_i\right)\right\rangle.$$

Any node u_j that receives subsequent messages from any other node, compares the received tuple $(u_{new}, dist_{new}, cost_{new})$ with its current tuple $(u_j, dist_j, cost_j)$. If $dist_j > dist_{new}$ then u_j discards its old tuple and stores the new one. If node u_j receives few broadcast messages with the same $dist_{new}$ then it selects a tuple for which $cost_{new}$ is smaller. In this way the overall cost of communicating with BS will be minimized [6].

The last step of building the communication tree is to inform the BS about all group keys K_i^G established by each CH. This is done through an onion message sent by each CH_i to its parent cluster head CH_j:

$$\mathrm{CH}_i \rightarrow \mathrm{CH}_j : O_1 = \left\langle u_i, E_{K_{i,BS}}\left(K_i^B\right)\right\rangle.$$

Upon receiving such message CH_j expands the message with its id u_j, encrypts it again and sends to the parent node:

$$\mathrm{CH}_j \rightarrow Parent(u_j) : O_2 = \left\langle u_j, E_{K_{j,BS}}\left(u_j, O_1\right)\right\rangle.$$

This messages are routed through tree structure up to the BS. This gives an information about communication tree structure and all group keys used in clusters. These keys will be latter used for verifying that routing tree is correct i.e. it leads to the BS.

Above procedure ensures that communication tree will be build securely if attackers cannot compromise nodes of the network. On the other hand, if attacker compromise the node, then he may broadcast fake messages with fake parameters – e.g. he may pretend to be a base station. It is very difficult to prevent such situation during building the communication tree but instead we propose a random verification procedure. In this manner each node u_i that stores a routing information $(u_i, dist_i, cost_i)$ with some probability asks BS to confirm his distance from the BS. This is done through an onion message sent to node's cluster head $\mathrm{CH}(u_i)$:

$$\begin{aligned} Node_{u_i} \quad &\rightarrow \mathrm{CH}(u_i) : \\ &O_1 = \left\langle u_i, E_{K_{BS,i}}(u_i, dist_i, nonce)\right\rangle. \end{aligned}$$

When receiving such message the CH adds its identifier u_j, its distance to the BS $dist_j$ and encrypts everything again with key $K_{BS,j}$:

$$Node_{u_j} \rightarrow \mathrm{CH}(u_j) : O_2 = \left\langle u_j, E_{K_{BS,j}}(u_j, dist_j, O_1)\right\rangle.$$

Successive CHs do the same procedure creating the onion message O_n. Upon receiving O_n BS decrypts and verifies whether number of encryption n is the same as $dist_i$. If so then it response with a confirmation:

$$BS \rightarrow Node_{u_i} : \left\langle E_{K_{BS,i}}(nonce + 1)\right\rangle.$$

On the other hand the BS detects that at least one node in the routing path is cheating and impersonates himself. When there is only one such node then the BS knows its position in the tree which equals $n - dist_i$. In this way the BS is capable of detecting compromised nodes.

4.4. Fault tolerance

When nodes fail to operate correctly (e.g. due to power depletion) then it may happened that tree structure breaks into distinct parts. This may cause a number of problems depending on whether node was a RN or a CH. If the node was a RN then neighbour nodes will eventually adjust their roles and takeover the tasks of the depleted one. In former case the communication tree becomes divided into two distinct parts and nodes in a subtree become orphans.

In such situation first nodes from the subtree have to detect that they are orphans and cannot communicate with the BS. This can be done in similar way as routing verification. If no response from the BS is received then node knows that become an orphan and has to reconnect the orphaned subtree to the main network. The way it happens is that each orphan u_i transmits to its cluster head CH_j a message to confirm its operation. When receiving such message CH_j response to the orphan u_i and asks its parent CH to confirm its operation too. This procedure is repeated by each CH that receives a request up to some CH_l that will not get a response since its parent CH_k depleted. In such situation CH_l, which shares a key with CH_k, broadcast a message to CH_k neighbours with an information that communication tree requires to be reconnected

$$\mathrm{CH}_l \rightarrow Nieghbours : \left\langle u_l, E_{K_k^G}(u_l, \text{"}reconnect\text{"})\right\rangle.$$

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Upon receiving such message all nodes in the neighbourhood repeat the clustering process. Afterwards new elected CH broadcasts a message indicating that it is looking for a parent:

$$\text{CH}_{new} \rightarrow Nighbours : \langle u_{new}, "orphan" \rangle .$$

All neighbour CH_i that receive that message response with:

$$\text{CH}_i \rightarrow \text{CH}_{new} : \left\langle u_i, E_{K_{new,i}}(u_i, dist_i, cost_i) \right\rangle .$$

Based on that messages new cluster head can verify whether node u_i is a legitimate node (i.e. knows encryptions key) and select a parent that minimizes routing costs to the BS. The last step is to generate new group key K^G_{new} and reroute the subtree if distances from the BS have changed.

4.5. Communication

In previous sections we have presented how nodes are capable of establishing encryption keys for one to one communication (pairwise keys) and group communication (group keys). These keys allow to ensure security in one-to-one and one-to-many communication. They also allow to authenticate one-to-one communication but fail to guarantee the originality of messages sent from the BS in broadcast mode (one-to-many communication). This allows a malicious CH to use the group key and send forged messages to neighbour nodes pretending that messages came from the BS.

Dimitriou [8] proposed to use one-way key chain commitments to defend against such impersonation. Nevertheless his proposition increase immunity it cannot defend against an adversary who may jam communication so that some nodes miss some messages and thus commitments. Afterwards adversary introduce new messages by "recycling" the unused commitments. As presented in [8] in such situation it will be impossible for nodes to notice the faked messages.

Instead of one-way keys we propose to solve this problem using commitments on messages sent by the BS so that each node will be able to verify that massages came from the BS in fact. To do this the BS has to extend each broadcasted message m with commitment computed as follows:

- BS computes $v = \text{HASH}(m)$ and $g_{BSv} = f(0, v)$,
- BS computes $w = f^{-1}(0, v) \bmod (p - 1)$ and $\alpha^w \bmod p$,
- BS selects a set Λ of t pairs of the form $(x_j, \alpha^{w g_{BSx_j}} \bmod p\rangle$ for random x_j such that for any node $u_i \neq x_j$,
- the commitment for message m has a form:

$$Commit = \langle \alpha^w \bmod p, \Lambda \rangle .$$

Upon receiving the message m and commitment each node u_i can verify its correctness by adding a pair $\left\langle u_i, \alpha^{w h_{u_i}(BS)} \bmod p \right\rangle$ to the set Λ and checking whether:

$$\alpha = \prod_{i \in \Lambda} \alpha^{w g_{BSx} \prod_{j \in \Lambda, j \neq i} \frac{v - x_j}{x_i - x_j}} \bmod p \qquad (6)$$

If (6) holds than node u_i is ensured that message m was sent by the BS. This is due to fact that only BS knows polynomial $f(x, y)$. Additional advantage of this authentication is that commitment depends on message being sent. Therefore an adversary cannot introduce new messages by "recycling" unused commitments as it was in [8].

5. Security Analysis

Proposed protocol for secure communication in ad hoc networks is based on secret sharing proposed by Shamir [18]. Shamir's proposal is t-resilient which means that it is secure as long as less then $t + 1$ users collude in order to compromise the security. Precisely, secret sharing assumes that at least $t + 1$ users have to use their shares in order to reconstruct the secret. In our proposal the function $f(x, y)$ is a secret that is shared among all nodes of the network. Each node poses the share which consist of two polynomials of degree t resulting from intersections of the surface $f(x, y)$ with planes $x = u$ and $y = u$. Therefore in order to describe unambiguously the whole surface $t + 1$ different polynomials are needed (either $g_u(x)$ or $h_u(y)$). Therefore our proposal is a (t, n) secret sharing scheme which is t-resilient – secure as long as less then $t + 1$ nodes are compromised (i.e. $t + 1$ polynomials are revealed).

Broadcast encryption was first proposed by Fiat and Naor [9]. Nowadays almost all broadcast encryption schemes are based on secret sharing and so is our proposal. Additionally we utilize random integers r and hide polynomial in the exponent modulo prime p. Both techniques enchants security and are widely described in the literature on broadcast exclusion (e.g. [13, 20]). As it was presented in [20] using random integers r ensures security for an unlimited number of executions of broadcast protocol. On the other hand we do not cope with traitor tracing and lack of anonymity of excluded nodes. Traitor tracing is out of scope of this paper while lack of the anonymity is an advantage since when nodes are excluded every other node immediately knows their IDs.

We now analyze the security ensured by the proposed scheme by discussing the general attacks on wireless ad hoc networks [12]:

- Spoofed, alerted or replayed routing information. This is the most direct attack against the routing protocol that targets information exchanged between nodes.

However, this is not a problem in proposed routing protocol since all information is encrypted and nodes posses an opportunity to verify the routing information securely.

- Selective forwarding. In this kind of attack an adversary is included in the communication tree (i.e. some cluster head has been captured) and selectively forwards some packets while drops the rest. This is a very difficult problem however our proposal increase security enabling nodes to verify the routing tree securely. Moreover, since all communication is encrypted the malicious CH has no information which messages it forwards or drops. Additional one-time passwords can be used to allow RNs to verify if they receive all communication from the BS. If some communication is missed than this indicates a possible selective forwarding attack.

- Sinkhole attacks. These kinds of attacks work by making a compromised node look especially attractive to neighbour nodes with respect to the routing algorithms. However, this kind of attack is not a problem in our proposal since routing information can be verified securely based on the secret key shared by each node and the BS.

- Sybil attack. This attack assumes that one node can present multiple identities to its neighbours thus appearing to be in more than one place in the network. By doing so, an adversary may be able to control communication in different parts of the routing tree. In our protocol this kind of attack does not apply since each node u_i is equipped with exactly one pair of secret polynomials $g_{u_i}(x)$ and $h_{u_i}(y)$. This information is not enough to present multiple identities.

- Impersonation attack. In this attack malicious node tries to impersonate itself as some other node or the BS. In out protocol this kind of attack does not apply to the BS since all communication from the BS is authenticated using broadcast encryption. On the other hand malicious CH may impersonate himself as one of its parents.

Security of the proposed protocol is solely based on the secrecy of polynomials $g_{u_i}(x)$ and $h_{u_i}(y)$ stored in nodes. These polynomials are of degree t and each have exactly one intersection point with BS's polynomials $g_0(x)$ and $h_0(y)$ which are also of degree t. Thus to break the security of the proposed protocol it is enough to compromise $t+1$ nodes from the network and use their information to reconstruct $g_0(x)$ and $h_0(y)$. Therefore t is a security parameter that should be chosen with caution: on one hand it increase security while at the same time increase communication and computation overhead - sizes of broadcast encryption messages and number of exponentiation in broadcast exclusion protocol depends linearly on t. For comparison to break the protocol proposed by Dimitriou [8] it is enough to compromise only one node from the whole network.

When selecting the parameter t during preprocessing phase one should take into account following properties:

- t should not be larger than the average number of RNs in one cluster. If t is larger than number of RNs then broadcast exclusion requires message of bigger length than sum of lengths of massages sent to each RN when one-to-one communication is used.

- t cannot be equal 1. If $t = 1$ then our proposal is the same as [8] with lack of authentication of BS's messages and without possibility to verify the routing information.

- t indicates the number of nodes that has to be compromised by an attacker to break the security of the proposed scheme.

Comparing our proposal to other solutions we should mention the TESLA protocol [17] which has good security properties. TESLA achieves authenticated broadcast communication by using on-way key chains and delayed key disclosure. TESLA, however requires loose time synchronization and cannot be applied to the communication trees.

6. Protocol Overhead

Proposed protocol ensures security and message authentication at similar cost as other protocols presented so far [8, 17]. Additional properties like broadcast exclusion and BS authentication require additional tradeoff in computation. This is mainly due to fact that security of broadcast exclusion is based on discreet logarithm problem.

On one hand this ensures security and allows to minimize network communication on group key exchange, rekeying and network management. A performance analysis of other protocols [8] and analysis presented in [1, 3] shows that the best GDH-like protocol requires number of messages proportional to the number of nodes in the cluster. For broadcast encryption protocol implemented in our proposal this overhead is of order $O(t)$ where t is a parameter chosen during preprocessing phase.

On the other hand broadcast encryption also requires nodes to compute exponents which is a complex operation for constrained devices. Nevertheless this inconvenience we think that our proposal is a promising opportunity for existing protocols. Moreover according to [11] each bit transmitted consumes as much power as 800-1000 instructions. Therefore we think that number of computation can

increase slightly if we decrease communication overhead at the same time.

Our proposal also requires additional storage at nodes to store secret data – polynomials $g_{u_i}(x)$ and $h_{u_i}(y)$. This requires $O(2t)$ memory but we think that it is allowed since memories become cheaper and smaller every year, creating a possibility to extend capabilities of nodes. Moreover this storage overhead is comparable to the one of GKMPAN-like protocols [23].

Proposed protocol requires additional preprocessing phase in which surface $z = f(x, y)$ is created. As mentioned earlier intersection of this surface with planes $x = u$ and $y = u$ results in two polynomials of degree t. Such property is achieved always when surface is of the form:

$$z = f(x, y) = \sum_{i=0}^{t} a_i(y)x^i = \sum_{i=1}^{t} \left(\sum_{j=0}^{t} a_{ij}y^i \right) x^i \quad (7)$$

According to (7) one has to choose t polynomials of variable y that will determine coefficients for polynomial of variable x. It can be also seen that if we put x or y equal constant value u then we always get a polynomial of degree t. Since exact equation for the surface (7) has to be known only to the BS thus no additional storage is required at nodes. We also assume that the BS is capable of storing all data needed to reconstruct surface's equation.

It is also worth to mention that in general degrees of node's polynomials $\deg(g_u(x))$ and $\deg(h_u(x))$ can be different – t and s respectively. From the security point of view however only the smallest value of those parameter matters. Therefore in our paper we assumed that those parameters are equal $\deg(g_u(x)) = \deg(h_u(x)) = t$.

7. Conclusions and Future Work

We have presented novel protocol for secure establishment of communication trees in ad hoc networks. Our protocol assumes that nodes are capable to store $O(2t)$ secret data and are capable of performing exponentiation. At those costs our proposal achieves high level of security ensuring:

- secure one-to-one communication encrypted with pairwise secret keys shared by each pair of nodes,
- secure group key exchange protocol based on broadcast encryption scheme,
- secure one-to-many communication encrypted with group keys,
- possibility to verify routing information and detecting orphans,
- possibility to verify authentication of messages sent by the BS,
- possibility to exclude up to t nodes within one cluster from future broadcast communication.

Comparing to other protocols from literature our proposal increase immunity to the coalitions of malicious nodes. In our protocol more than t nodes have to collude in order to reconstruct surface equation $f(x, y)$ and break the scheme. Another advantage is low communication overhead on group key establishment which equals $O(t)$ per each cluster.

Our future work will attempt to adjust proposed protocol for mobile ad hoc networks (e.g. Vehicle Ad Hoc Networks – VANETs).

References

[1] Y. Amir, Y. Kim, C. Nita-Rotaru, G. Tsudik: The Performance of Group Key Agreement Protocols, Proceedings of the 22nd IEEE International Conference on Distributed Computing Systems, Viena, Austria, June 2002

[2] E. Anton, O. Duarte: Group key establishment in wireless ad hoc networks, Workshop on Quality of Service and Mobility (WQoSM), 2002

[3] E. Anton, O. Duarte: Performance Analysis of Group Key Establishment Protocols in Ad Hoc Networks, Technical report, Universidade Federal de Rio de Janeiro, Brazil, 2006

[4] S. Buchegger, J.-Y. Le Boudec: Cooperative Routing in Mobile Ad Hoc Networks: Current Efforts Against Malice and Selfishness, Mobile Internet Workshop, Informatik 2002, Dortmund, Germany, LNCS, 2002. Springer

[5] C. Castelluccia, N. Saxena, J. H. Yi: Self-configurable Key Pre-distribution in Mobile Ad Hoc Networks, LNCS, vol.3462, pp.1083-1095, 2005

[6] Z.Chaczko, R.Klempous, J.Nikodem, M.Nikodem, J.Rozenblit: An improvement of energy aware routing in wireless sensors network, 2-*nd* European Modeling and Simulation Symposium, EMSS 2006, Barcelona, Spain, October 4-6, 2006

[7] R. Curtmola, S. Kamara: A Mechanism for Communication-Efficient Broadcast Encryption over Wireless Ad Hoc Networks, 2nd Workshop on Cryptography for Ad hoc Networks, Venice, Italy, July 16th, 2006

[8] T.D. Dimitriou: Securing Communication Trees in Sensor Networks, ALGOSENSORS 2006, to appear in LNCS, Springer-Verlag, 2006

[9] A. Fiat, M. Naor: Broadcast Encryption, CRYPTO'93, LNCS, vol.773, pp.480-491, Springer-Verlag, 1994

[10] A. Filipovic, A. Datta, C. McDonald: Low-Latency Routing in Ad Hoc Wireless Networks Using Finite Horizons, LNCS, vol.3759, pp.422-433, 2005

[11] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler and K. Pister: System architecture directions for networked sensors, Proceedings of ACM ASPLOSIX, November 2000

[12] C. Karlof, D.Wagner, Secure Routing in Wireless Sensor Networks: Attacks and Countermeasures, Elsevier's Ad Hoc Network Journal, Special Issue on Sensor Network Application and Protocols, vol.1, pp.293-315, September, 2003

[13] C.H. Kim, Y.H. Hwang P.J. Lee: Practical Pay-TV Scheme using Traitor Tracing Scheme for Multiple Channels, WISA'2004, LNCS, vol.3325, pp.265-279, Springer-Verlag, 2004.

[14] R. Kui, Y. Gang Yao: Key Agreements in Ad Hoc Networks, In Proc. of Advance in Cryptology, Chinacrypt04, Shanghai, 2004

[15] H. K. Lee, H. S. Lee, Y. R. Lee: An Authenticated Group Key Agreement Protocol on Braid groups, eprint 2003/018.

[16] D. Liu, P. Ning, K. Sun: Efficient Self-Healing Group Key Distribution with Revocation Capability. In Proc. of the 10th ACM Conference on Computer and Communications Security (CCS '03), 2003

[17] A. Perrig, R. Canetti, J.D. Tygar, D. Song: Efficient Authentication and Signing of Multicast Streams over Lossy Channels, IEEE Symposium on Security and Privacy, pp.56-73, 2000

[18] A. Shamir, How to Share a Secret, Communications of the ACM, vol.22, no.11, ACM Press, New York, NY, 1979

[19] M. Steiner, G. Tsudik, M. Waidner: Diffie-Hellman Key Distribution Extended to Group Communication, ACM Conference on Computer and Communications Security, pp.31-37, 1996

[20] Y.Watanabe, M.Numao: Multi-round Secure Light-Weight Broadcast Exclusion Protocol with Preprocessing, LNCS, vol.2808, pp.85-99, 2003

[21] F. Ye, A. Chen, S. Lu, L. Zhang: A Scalable Solution to Minimum Cost Forwarding in Large Sensor Networks, 10th International Conference on Computer Communication and Networks, pp. 304–309, 2001

[22] O.Younis, S.Fahmy: HEED: A Hybrid, Energy-Efficient, Distributed Clustering Approach for Ad Hoc Sensor Networks, IEEE Tans.on Mobile Computing, vol.3, no.4, October-December 2004

[23] S.Zhu, S.Setia, S.Xu, S.Jajodia: GKMPAN: An Efficient Group Rekeying Scheme for Secure Multicast in Ad Hoc Networks, Technical report ISE-TR-02-01, 2004

AUTHOR BIOGRAPHIES

JAN NIKODEM received his M.Sc. in Artificial Intelligence in 1979 and Ph.D. degree in Computer Science in 1982 from Wrocław University of Technology (WUT), Poland. Since 1986, he has been an Assistant Professor in the Institute of Technical Cybernetics, WUT. Since 2005 in the Institute of Computer Engineering, Control and Robotics. His current research are focused on the area of wireless sensor networks, digital data transmission protocols and embedded systems.

Maciej NIKODEM received his MSc degree in Computer Science from Wrocław University of Technology in 2002, now he is on Ph.D. studies in Computer Science. His research interests include cryptography, cryptanalysis, wireless communication and data base security

A Multi-Tier, Multi-Role Security Framework for E-Commerce Systems

Dr. Ernest Cachia, Mr. Mark Micallef
Software Engineering Process Improvement Research Group
Department of Computer Science and Artificial Intelligence
University of Malta
ernest.cachia@um.edu.mt, mmica01@um.edu.mt

Abstract

As the use of the internet for commercial purposes continues to grow, so do the number of security threats which attempt to disrupt online systems[1][8][9]. A number of these threats are in fact unintended[12]. For example, a careless employee might drop a cup of coffee onto essential equipment. However, when compared to the brick and mortar world, the internet offers would-be attackers a more anonymous environment in which to operate. Also, the free availability of hacking tools makes it possible even for the curious teenager to carry out dangerous attacks.[3]. Despite this ever-present threat however, it is all too often the case that security is dealt with (if at all) after a web application has been developed[2]. This is mainly due to our software development heritage whereby companies prefer to focus on the functionality of new systems because that provides and immediate return on investment.

This paper proposes a framework for building security into web applications as they are being developed. The core philosophy here is that security is too big an issue to leave up to one person/team after the product has been developed. The framework also provides a quality assurance process and a communication protocol to ensure that all security-related tasks have been carried out.

Keywords: *Security, Software Quality Assurance, Web Applications, E-Commerce*

1. Introduction

W. Edwards Deming stated several years ago that the quality of a product is directly related to the quality of the process used to create it"[4]. Although Deming's work involved production work during World War II, his statement holds true to this day today when dealing with complex software systems. As part of a wider body of work dealing with the rapid development of high-quality e-commerce systems[6], the authors of this paper identified the five most important quality attributes in e-commerce systems of which the most important was deemed to be security[5].

Despite the importance of security, it is still often the case that high-profile breaches surface in the news with many more going unannounced[3][8][9]. It appears that development companies focus mostly on functionality when developing a system since this is perceived to provide an immediate return on investment. However, a study amongst 350 online shoppers, by the authors of this paper revealed that 86% of potential customers would not shop at a site if they were not confident in its security capabilities[5]. Also, 36% of online shoppers consider security considerations as the primary reason for choosing a brick and mortar store over an online equivalent[5]. This leads to the observation that web application developers are not really delivering a product of good quality when they concentrate on functionality, but rather they deliver one of perceived quality. The lack of focus on security throughout a web application's development life cycle often leads to a vulnerable first release of that application[2]. Considering potential customers' security awareness, this may prove to be a costly mistake. This paper explores the world of online security threats and proposes a process-based framework for their prevention. As explained in section 2, the framework splits the building of web application embedded security into digestible chunks. A clear benefit of this is that the responsibility of security is shared by multiple players who participate in a web application's development life cycle.

2. Principles behind the paper

The proposed framework is based on three fundamental principles.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Firstly, security is a quality attribute which like other quality attributes, should be built into the system throughout its life cycle and not added on as the system nears completion[10][11]. There are a number of popular misconceptions that securing a web application simply involves setting up a firewall and/or using secure connections such as SSL[12] between the clients and servers. However, with increasingly sophisticated attacks being developed and the multi-faceted nature of security, it is felt that such approaches will contribute towards solving the security problem but will not eliminate it. A system is only as secure as its weakest link[14] and as such, the framework provides a routine way in which security can be built in at different stages of development. One could also argue that testing for security after an application has been developed is likely to lead to a vicious circle of testing, patching and consequent degradation of system functionality[15]. Such approaches usually have a negative affect on the system's maintainability[16] and the question of "did we miss anything?" is likely to linger on even after release since no one can guarantee that all the security problems had been discovered and fixed.

The second principle on which this framework is based involves the sharing of responsibility for security by all the parties contributing to the development and deployment of a web application. By having everyone build in their own aspect of security, securing the web application becomes a more structured and hence controllable effort. Also, the resultant system is likely to be more secure as different security viewpoints are built in by people who are most knowledgeable about that particular aspect of security. For example, a software developer will and it relatively easy to make sure that all input fields in an online form are validated (unvalidated input fields are a top security risk in web applications [17]) where as a security engineer who is responsible for all security issues related to a web application product may have difficulties verifying such validation. Conversely, the security engineer would feel comfortable analysing system logs to detect potential intrusions whereas a developer might find this to be a daunting task.

The third and final principle acknowledges the dynamic nature of security issues. Although it is believed that sound engineering practices will eliminate most security problems, it would be unrealistic to claim that one can guarantee protection against future threats. Therefore, the proposed framework provides methods through which new countermeasures can be constructed by users of the framework in order counteract future security treats.

3. Important Issues

This section discusses a number of issues which are required to understand the way the security framework is designed to operate. More specifically, this section outlines issues relating to the different types of security threats, software development life cycle concerns and typical roles taken by contributors to a web application's development life cycle.

3.1. Types of Security Threats

Through research in psychology, it has been established that humans are able to handle large quantities of information much more efficiently if they are able to classify it into a manageable number of categories[18]. As a precursor to this work, the authors of this paper developed an ontology of security threats which pertain to web applications[7]. Within this ontology, five broad types of security threats were identified. These are shown in the figure 1.

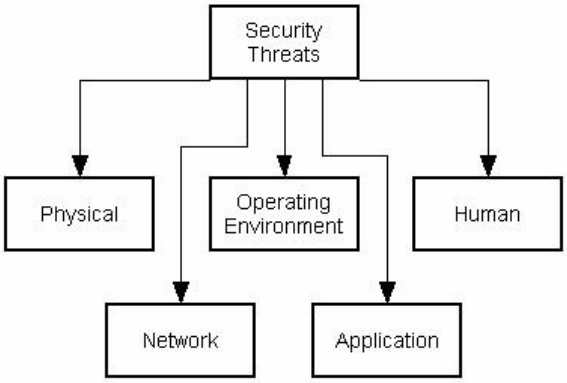


Figure 1 - Types of security threats

A web application may be compromised by threats which exploit vulnerabilities relating to any one of the categories shown in figure 1[7]. All too often, one may be lulled into a false sense of security by emphasising security against threats on one of the above-mentioned levels. For example, one could feel that using secure communications[13] is enough to ensure a particular application's security. However, secure communications only assure privacy at the network level and a private web conversation with a hacker will not deter him/her from exploiting vulnerabilities of a different nature. The following is a brief discussion of the threat categories (sometimes referred to as levels) shown in figure 1.

At the ***physical level***, one must ensure that the locations where hardware (be it for deployment or development purposes) resides are physically

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

secured[19]. Any physical intrusion into such a location may resulting interruption of service, stolen data, and so on. In such cases, intruders may not necessarily be outsiders but could also be insiders (e.g. disgruntled employees).

Another category of security threats involves exploiting vulnerabilities at the ***network*** level. Such threats attempt to manipulate shortcomings in communication protocols to disrupt service and/or gain access to private information. Appropriate actions must be taken to prevent this from happening[12].

The ***operating environment*** where a web application is hosted is also vulnerable to attacks. Vulnerabilities may be present in any components ranging from the operating system [20] to any other web-enabled component such as web server software, mail server software, and so on[21]. Again, one can easily imagine a scenario where an application is highly secured by developers, only to be breached by attacking its operating system instead of the application itself.

The category of ***application level*** threats refers to threats which exploit vulnerabilities within the web application itself. SQL injection attacks, cross-site scripting and authentication-related attacks are typical examples of threats which can be found at this level[17][22]. It is beyond the scope of this paper to explore the technical merits of such attacks.

Finally, one should discuss vulnerabilities at the ***human level*** of security. Such vulnerabilities involve trusted human beings knowingly or unknowingly enabling outsiders to breach a web application and access restricted data[23]. Numerous incidents have been reported whereby a technically secure web application was breached because an insider was tricked into revealing information such as their username and password (a technique commonly known as social engineering). An effective security policy must ensure that trusted users are in a position to repel any attempts made to use them as a weak point within the application's security structure.

3.2. Lifecycle Issues

In another publication by the authors of this paper[6], the argument was made that based on a number of surveys and scientific research, the nature of e-commerce systems and the time frames in which they are developed makes them ideal candidates for a RAD-based life cycle. The authors also proposed a RAD-based life cycle which is specifically tailored to such systems[6]. To this extent, the framework proposed here is designed to be integrated into a RAD-based development life cycle such as DSDM[24].

3.3. Roles within a Web Application Development Team

Throughout its lifetime, from conception to deployment, a web application will be influenced by a number of people, all of whom have one or more roles to play with respect to that application. As already discussed, the security framework proposed in this paper spreads the process of building security into a web application across the development life cycle as well as placing responsibility on a number of different contributors. Typical “roles” which influence an application's development usually include the software developer role, the tester role, the security engineer role, and so on. Depending on the size of the organisation in question, there may be multiple persons with the same role or even single people who take on the responsibility of more than one role throughout the application's life cycle.

The processes being proposed in this paper do not assume any particular configuration of human resources within a project. This is mainly due to the fact that different companies organise their human resources in different ways. For example, whilst some companies may have software engineers working on single projects carrying out analysis, design, development and testing, others may have specialists carry out specific tasks across a number of projects. It is beyond the scope of this framework to impose development roles on an organisation as this would immediately render the framework unusable by organisations who are not in a position to change their human resources structure. However, one must be aware of who does what within the organisation in order to use the methods being proposed in this framework.

4. Framework Components

Generically speaking, the proposed framework provides mechanisms for the prevention, detection, and recovery from security threats to web applications[12]. The main emphasis in the process is placed on prevention since it is more desirable to not suffer a security breach than it is to be able to recover from one. However, detection and recovery mechanisms should always be in place in case of a security threat managing to materialise.

Since the framework spreads responsibility for security across multiple stages and multiple players,

project managers need to be sure that everyone is actually living up to their obligations. Consequently, the framework proposed here provides a number of assurance and accountability mechanisms for this purpose. Also, effective communication between contributors to a web application is very important. In many cases, one person's task may depend on other tasks being completed. In the flurry of activity which is constantly happening during a development project, an underlying order in the way people communicate is essential. Therefore, the framework also proposes a simple communication protocol which keeps everyone up to date with the status of their relevant tasks.

Given the above considerations, the proposed framework is composed of four main components as follows: a Threat Profiling Process, a Threat Management Process, an Assurance & Accountability Process and a Team Communication Protocol. Figure 2 depicts how these components interact together and is followed by a detailed description of each one. In order to give the proposed framework a practical context, each section will give an example of how the framework would be used to deal with a real-world threat. The threat in question will be the threat of ***Unvalidated Input***.

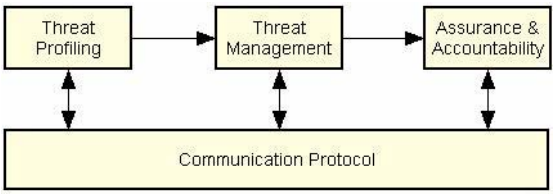


Figure 2 - Framework Components

4.1. Threat Profiling Process

This component of the framework is the most crucial of all and deals with identifying a new security threat and determining how it should be tackled. Based on reasoning in a precursor to this work by the authors of this paper [7], a threat profile consists of four categories of information as shown in figure 3.

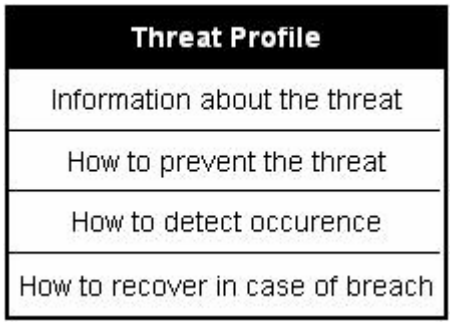


Figure 3 - A high-level threat profile

4.1.1. Information about the threat. This category of information deals with identifying details about the threat, what harm it can cause and which level of security (see section 3.1) it would breach. The following information should be collected about a threat:

- **ID** - Give the security threat an internal unique ID for later reference in the framework and possible future use of storing in a database.
- **Name** - A human-readable name which identities the threat.
- **Security Level** - The security level (section 3.1) which the threat pertains to.
- **Severity Level** - An indication of the amount of harm and/or fallout which would result from the threat if it materialises. It is proposed that this information be categorised as low, medium or high. A low severity level indicates that the consequences of a materialisation of this threat would have little repercussion on the web applications ability to keep functioning and on its reputation. An example of this might involve a relatively harmless adware program managing to install itself on a server. A threat with medium severity is one that causes a degradation in service without completely disabling the site or one that causes damage or loss which can be recovered. Examples might include minor website defacements, or loss of data which is regularly backed up and can thus be retrieved. Finally, a high severity level indicates threats which severely impinge on a web application's stability and/or its users' privacy and security. Examples include hackers gaining access to clients' personal information, viruses corrupting or erasing data in such a way that it cannot be retrieved, and so on.
- **Environments Affected** - Information about which environments are affected by this threat. This may include information about operating systems, web servers, firewalls, and so on.
- **Threat Description** - A sufficiently detailed description of what the threat does, which vulnerabilities it exploits, and the repercussions should the threat materialise.
- **People Involved** - A list of people/roles involved in potentially introducing and/or solving the problem.
- **Introduced In** - A list of life cycle phases in which the threat is likely to be introduced.

In terms of a practical example, the security information about the *Unvalidated Input* would look something like this. For the sake of brevity, some details are omitted from practical examples:

Threat Information	
ID:	001
Name:	Unvalidated Input
Security Level:	Application
Severity Level:	High
Environments Affected:	All browsers, all servers
Description:	Web applications use input from HTTP requests (and occasionally files) to determine how to respond. Attackers can tamper with any part of an HTTP request, including the URL, query string, headers, cookies, form fields, and so on. If all input to a web application is not properly validated before being processed, it can lead to situations where a buffer overflow or injection attack can occur.
People Involved:	Designers, Developers, Team Leads, Testers
Introduced in:	Design, Development

Figure 4 - Threat Information for Unvalidated Input

4.1.2. How to prevent the threat. This part of the proposed framework involves important work which influences the effectiveness with which the threat in question could be prevented. During this stage of the framework, representatives of each role in a team would meet and brainstorm how a threat could be prevented. Participants must work with three elements: the threat itself, the life cycle stages in which it can be prevented, and finally the people who will prevent it. No predefined series of steps for carrying out this step is being proposed at this stage. This is because each team has its own dynamics when conducting a brainstorming session and imposing a rigorous structure on such meetings may prove to be counter productive. However, two deliverables are required from this stage of the framework. The first is a prevention document which describes what steps need to be taken to prevent the threat from materialising, who will take them and at what stage of the development lifecycle. Depending on the practices and conventions employed by the organisation in question, such a document could be text-based or graphics-based. The practical example in figure 5 takes a graphical approach.

The second deliverable is an coverage and accountability matrix. This is a condensed version of the prevention document which will be combined with accountability matrices from other threats later on in the framework to ensure that all security threats have been catered for. The matrix will list the tasks which need to be carried out in row headings, and the people responsible for carrying them out in the column headings. An appropriate symbol representing a stage in the life cycle is then placed at the intersection of a row and column to signify that a particular person should carry out a particular task at a particular stage in the life cycle. An example of this is shown in figure 6.

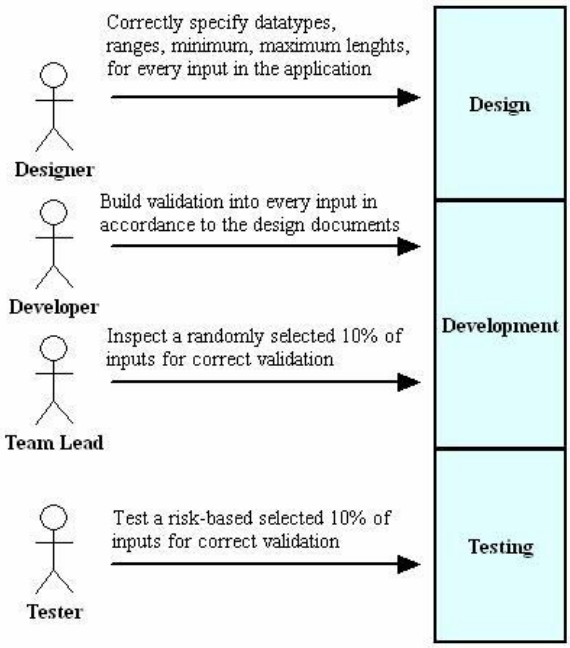


Figure 5 - An example of a threat-prevention document

	Designer	Developer	Lead	Tester
Specify all validation				
Develop validation code				
Inspect code (10%)				
Test code (10%)				

Testing Development Testing

Figure 6 - A typical accountability matrix

4.1.3. How to detect and recover from threat materialisation. Despite all the planning and prevention mechanisms, instances may arise whereby a particular threat still materialises. This may happen for various reasons, which include new security threats which have not yet been incorporated into an organisations processes, human error, and so on. Given the possibility of security breaches actually occuring, a process must be in place to detect a breach deal with it. The same brain storming session which deals with how to prevent the threat (section 4.1.2) should be extended to deal with detecting an occurrence of the threat and also define how to recover from it. Again, no rigorous structure is imposed here but the reaction to a security breach must include the following categories as a minimum:

- Implementation of detection mechanisms for the threat
- Immediate steps to minimise the effects of the breach (damage control)

- Identify and fix the cause of the breach
- Restore the web application to normal operation
- Review procedures which deal with that particular threat so as to minimise the risk of a future breach

Of course, the team may decide to include further steps in the case of particular breaches. In a practical context, a detection mechanism for threats related to unvalidated input may be the detection of a database error. If testing was properly carried out, a database error may be a strong indication that someone has gained access to the database and is trying to determine its structure. If it is established that an attack has in fact occurred, one process which could be implemented would be as follows:

1. Identify and disable the section of the site which allowed the materialisation.
2. Identify damage which was cause and attempt to reverse it.
3. Develop, test and deploy a fix
4. Analyse why the threat materialised and who was responsible for it
5. Take corrective action to minimise future occurrences (e.g. train developers, modify procedures, etc)

4.2. Threat Management Process

Within the proposed framework, once a threat has gone through the threat profiling process, it should be integrated into the threat management process. This process consists of the following three steps. The first step involve composing together all Accountability and Coverage matrices. This will result in a matrix which displays information about all security-related actions which must be taken by different people at different stages of the application's life cycle. Once this has been done, the newly composed matrix is broken up into several lists of tasks for each person involved with the application. Tasks are grouped together in categories corresponding to the stage of the life cycle in which they should be carried out. This is a simple mechanical process which could easily be automated through a computer program. At this stage, each person working on an application will receive a task-list which is tailored to his/her role within the application. Figure 7 gives a simplified example of two different people receiving customised task lists.

The final stage involves each person making sure that all his/her assigned tasks have been carried out as each phase of the life cycle comes to an end. Should someone have difficulty understanding what exactly needs to be done, he/she could refer to the relevant prevention document which was produced in the threat profiling stage (section 4.1.2).

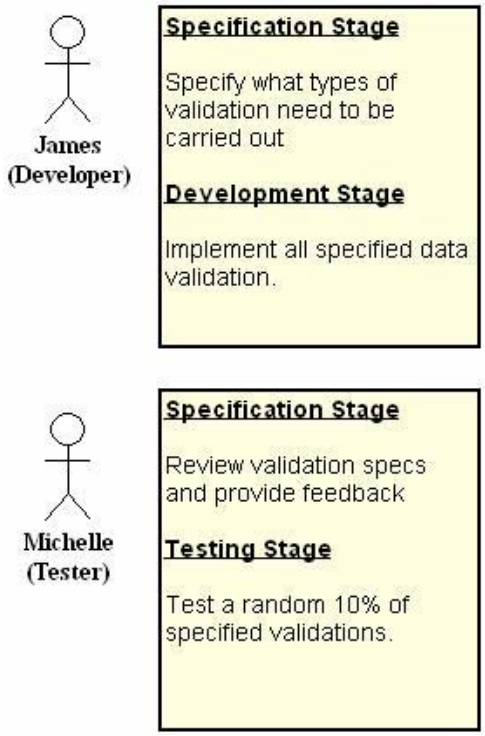


Figure 7 - An example of personalised task lists

4.3. Assurance and Accountability Mechanisms

In an ideal world, human beings would be infallible. However, we do not live in such a world. People have the tendency to make mistakes, forget to perform tasks, or even intentionally avoid carrying out all the work they are assigned to do. If a developer forgets to implement validation checks on a web application's inputs, then it becomes vulnerable to a number of attacks, thus rendering everyone else' efforts to secure the application effectively futile. In view of this, the proposed security framework provides the following assurance and accountability mechanisms.

Firstly, each person on the team will be given a tick-list which is basically a copy of the task list with a tick-box next to each task. As each task is completed, one would tick the associated box in the tick-list. As a fundamental requirement, each person is required to sign off a completed tick-list at the end of each stage of the life cycle. Much in the same way that agile techniques such as DSDM[24] recommend pairing developers up to work on the same tasks, it is being recommended that people pair up to verify that each other's security responsibilities have been met. That is to say, each partner walks the other through the

security related actions which were taken during a particular stage of the development life cycle. This need not be a complicated or time consuming task but rather an informal walkthrough of what was done. Since other checking mechanisms are likely to have been put in place during the brain storming session (section 4.1.2), this pairing up is an optional recommendation which although beneficial, could be perceived to be a strain on a small organisation's resources.

4.4. Communication Protocol

The final component of the framework deals with communication between team members. In many instances, a particular security threat may need to be counteracted by more than one person within a team. This is in fact one of the principles behind this framework. In such cases, communication between the different team members is key because the work of one person could easily undo the work of another.

Consequently, a communication protocol is being proposed whereby team members send predefined messages to each other at key stages related to their work. For each threat, a series of communication messages are defined. In some cases, messages may be as trivial as notification that a particular task has been carried out whilst other messages may include more information such as detailed specifications. A message consists of a trigger, a source, a list of recipients and some form of content. The trigger is an event which triggers the sending of the message (e.g. a task being completed) whilst the content could be anything from a simple sentence to a complex document.

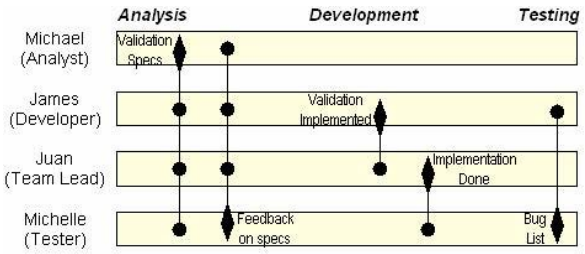


Figure 8 - An example of messages relating to particular threats

Consider the example shown in figure 8. This diagram depicts communication between different people during different stages of the life cycle. A message is represented by a line with a diamond shape indicating the originator of the message and a circle for each recipient. The first message is a specification document originating from the analyst to the other three people involved. Following that, the tester sends feedback about the document to the team lead, developer, and analyst. During the development phase, the developer sends a notification to the team lead that the validation has been implemented, who in turn informs the tester that all development has been completed. Finally, the tester tests the implementation and sends a list of discovered problems to the developer.

Once it has been effectively specified, required communication becomes a routine task which can be listed on a team member's task list and hence significantly reduce the chances of someone forgetting to relay important information to other parties.

5. Conculsions and Future Work

It is believed that the framework presented here provides a fully comprehensive process for building security into web applications. The multi-tier, multi-role approach seeks to mitigate the overwhelming task of securing a web application by distributing the responsibility for security amongst all contributors to the development of the application.

Future work in this area will involve using the processes proposed here to create threat profiles for a sufficiently large number of security threats so as to make the framework useful in practical setting. Once this has been done, an industry trial will be performed whereby the framework will be used during a number of development projects in order to verify the effectiveness of the methods presented in this paper.

6. References

List and number all bibliographical references in 9-point Times, single-spaced, at the end of your paper. When referenced in the text, enclose the citation number in square brackets, for example [1]. Where appropriate, include the name(s) of editors of referenced books.

[1] Glisson W. B., Welland, R., “Web Development Evolution: The Assimilation of Web Engineering Security”, Proceedings of the 3rd Latin American Web Congress, IEEE Press, 2005.

[2] Gaur N., “Assessing the Security of Your Web Applications”, Linux Journal, April 2000

[3] Morrisdale P.A., “The Six Dumb est Ideas in Computer Security"”, http://www.ranum.com/security/ computer security/editorials/dumb/

[4] Rakitin S. R., “Software Verification and Validation: A practitioner's Guide”, Boston: Artech House, 1997

[5] Cachia E., Micallef M., “Towards Effectively Appraising Online Stores”, Proceeding of the Software Engineering Knowledge Engineering (SEKE) Conference, 2004

[6] Cachia E., Micallef M., “Towards a RAD Framework for E-Commerce Systems”, International Conference on Web Information Systems and Technologies, 2006

[7] Cachia E., Micallef M., “An Ontology of Threats to Web Applications”, Proceedings of the Computer Science Annual Research Workshop, 2006

[8] Deloitte, “2005 Global Security Survey", Deloitte Touche Tohmatsu, 2005

[9] Gordon L. A., Loeb M. P., et al, “2005 CSI/FBI Computer Crime Survey”, Computer Security Institute, 2005

[10] Hersherb J., et al “Software quality and the Capability Maturity Model”, Communications of the ACM, June 1997

[11] Pressman R. S., “Software Engineering, A Practitioner's Approach”, McGraw Hill, 2000

[12] Mackey D., “Web Security for Network and System Administrators”, Thomson Course Technology, 2003

[13] Freier A. O., Karlton P., Kocher P. C., “The SSL Protocol Version 3.0”, Netscape, 1996

[14] Schneider B., “Secrets & Lies: Digital Security in a Networked World”, John Wiley & Sons Inc, 2000

[15] Lehman M. M. et al, “Metrics and Laws of Software Evolution - The Nineties View”, Fourth International Software Metrics Symposium (METRICS'97), 1997

[16] Baldwin J., Coady Y., “Are patches cutting it?: structuring distribution within a JVM using aspects”, Proceedings of the 2005 conference of the Centre for Advanced Studies on Collaborative research, IBM Press, 2005

[17] “The Open Web Application Security Project”, http://www.owasp.org/

[18] Antherton J. S., “Learning and Teaching: Assimilation and Accommodation” http://www.learningandteaching.info/learning/assimacc.htm ,2005

[19] Diroff T.E., “The protection of computer facilities and equipment: physical security”, ACM SIGMIS Database, ACM, 1978

[20] Blakely R., “Hackers Uncover Biggest Microsoft Vulnerability”, TimeOnline, http://business.timesonline.co.uk/article/0,,9075-1968021,00.html, 2006

[21] Nagel B., “Microsoft Releases Out-of-Cycle Patch for VML Flaw”, http://www.redmondmag.com/news/article.asp?EditorialsID=7825, 2006

[22] Jovanovic N., Christopher K., Engin K., “Precise Alias Analysis for Static Detection of Web Application Vulnerabilities”, Proceedings of the ACM SIGPLAN Workshop on Programming Languages and Analysis for Security, 2006

[23] Orgill G. et al, “The Urgency for Effective User Privacy-education to Counter Social Engineering Attacks on Secure Computer Systems”, Proceedings of the 5th conference on Information technology education , ACM Press, 2004

[24] DSDM Consortium, “DSDM - Business Focused Development”, Addison-Wesley, 2003

Alert Fusion for A Computer Host Based Intrusion Detection System

Chuan Feng, Jianfeng Peng, Haiyan Qiao, Jerzy W. Rozenblit
Electrical and Computer Engineering Department
The University of Arizona
Tucson, Arizona 85721
Email: {fengc, jeff, haiyanq, jr}@ece.arizona.edu

Abstract

Intrusions impose tremendous threats to today's computer hosts. Intrusions using security breaches to achieve unauthorized access or misuse of critical information can have catastrophic consequences. To protect computer hosts from the increasing threat of intrusion, various kinds of Intrusion Detection Systems (IDSs) have been developed. The main disadvantages of current IDSs are a high false detection rate and the lack of post-intrusion decision support capability. To minimize these drawbacks, we propose an event-driven intrusion detection architecture which integrates Subject-Verb-Object (SVO) multi-point monitors and an impact analysis engine. Alert fusion and verification models are implemented to provide more reasonable intrusion information from incomplete, inconsistent or imprecise alerts acquired by SVO monitors. DEVS formalism is used to describe the model based design approach. Finally we use the DEVS-JAVA simulation tool to show the feasibility of the proposed system.

1 Introduction

The scale and intensity of information attacks have risen over the years. These attacks usually cause severe loss to individuals and businesses, and threaten the security of a nation. Routine authentication and access control mechanisms have proven inadequate in preventing attacks. Information security infrastructure has relled more on Intrusion Detection Systems (IDSs) [1]. Generally, an IDS detects unwanted manipulations to a system, such as network attacks against vulnerable services, data driven attacks on applications, host based attacks such as privilege escalation, unauthorized access to sensitive files, and malwares.

Many techniques have been applied within IDSs to detect intrusions, such as expert systems, state transitions, probabilistic approaches, process profiling, etc. These various IDSs can be categorized into two classes: signature-based and anomaly-based detections [2]. Signature-based IDSs compare current events with known attacks and look for similarities. The major limitation of the signature based detection method is that it cannot detect novel attacks. Anomaly based IDSs do not share this limitation since they model normal behaviors and attempt to identify abnormal activities of computer performance metrics. Fore this reason, anomaly detection exceeds signature based detection. However, anomaly detection systems are seldom implemented due to their high false alert rate. Our intention is to build a host based intrusion detection system that can detect novel attacks with a low false alert rate.

All IDSs are based on the belief that an intruder's behavior has measurable differences from that of a legitimate user. Therefore how to manipulate the incoming event is a big issue. Traditionally, IDSs treat an incoming event as a single signal, which increase both false positive and negative alert rates. Our solution is to apply Subject-Verb-Object multi-point monitors, which can detect deviations from normal behavior and trigger different alerts.

The issue of the SVO model is that when an IDS runs, it will generate a tremendous number of events in real time. Hence, it is inefficient and impractical for the system administrator to react promptly. To minimize the cognitive overload on administrators, yet present them with the information they require, an alert fusion model is implemented to collate the alerts from different sensors which provide inconsistent information [6] [8] [11] [13] .

The proposed fusion model employs multi-level processing architecture, which identifies categories of techniques and algorithms for performing the specific functions [3] [4]. In addition to the source data and model interface, four processing levels are involved: source preprocessing, alert data normalization, spacial alert fusion, and temporal alert fusion. A knowledge base is applied to improve the performance of the fusion processes. Each processing level has knowledge of the nature of the problem.

This paper is organized as following: section 2 describes the SVO model and system architecture of the proposed

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

IDS. Section 4 presents the multi-level alert fusion model. In section 3, we discuss the DEVS model design and simulation using DEVS-JAVA in detail.

2 Event-Driven Intrusion Detection System

2.1 SVO Model

Subject-Verb-Object (SVO) is a linguistic typology concept we use to describe the intrusion event. The Subject refers to the origin of the action; the Verb indicates the subject's action; and the Object denotes somebody or something involved in the subject's performance. Any event happening inside a computer has its subject, which most often is a running process. Here we explicitly differentiate the subject from the owner, who is the actual user that runs the process. Similarly, the verbs and objects can be identified. Putting the subject, verb and object into a triple allows us perform fine grain analysis of the events.

In order to reduce the triple events combination, a critical event set is defined so that events out of this set either pose no threat or are trivial threats that can be ignored. In the current stage of our research, we focus on the critical set that contains important files we want to protect.

2.2 System Architecture

With the SVO model and critical event set, the IDS architecture is shown in Fig 1. In the figure, there is an SVO anomaly detection engine, which consists of separate Subject, Verb and Object detection models; an alert fusion engine; and a decision support model. They address problems such as how to detect an abnormal event, what to do with an alert and how to assist the system administrator when a threat is identified.

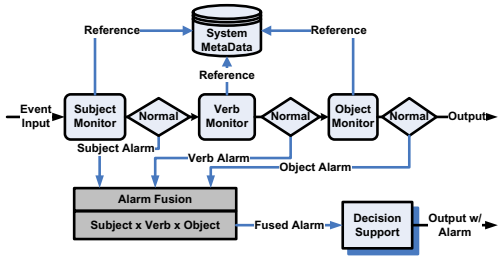


Figure 1. IDS system

3 Alert Fusion Model

The goal of the alert fusion model is to get more consistent intrusion information based on the alert data coming from the anomaly detection engine. The multi-level processing model of the fusion model is shown in Fig. 2.

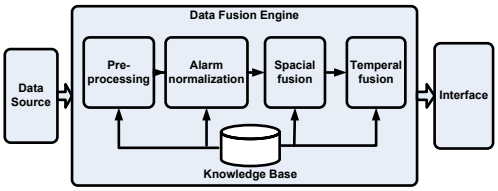


Figure 2. Alert fusion model

The Data Source at the left end of Fig. 2 indicates alerts from different anomaly monitors. The right end of the figure shows the system interface to the fusion system. Four processing levels are presented in the figure. The first one is Source Preprocessing, which synchronizes the information flow from different sensors and reduces data overhead for further levels. The second processing level is Alert Normalization, which transforms different alerts into a consistent set of scale. The third level is Spacial Alert Fusion, which fuses alerts from different anomaly detection monitors. The fourth level is Temporal Alert Fusion, which combines alerts within a time sequence and gives more useful intrusion information. The four levels are described below in greater detail.

3.1 Source Preprocessing level

Source Preprocessing, the first step of alert fusion, addresses reducing the overhead alert information and distributing data to appropriate processes.

Following the SVO model, an event, such as "process A read file B" will be divided into separate parts for analysis. Because each anomaly detection monitor has its own operational delay, the detected information (Alert/Normal) is sent into the fusion model asynchronously. Before any further operations, the preprocessing level synchronizes the information flow. To discern events, a unique EventID is assigned to each event before it is brought into various monitors. Our synchronization engine uses the EventID to combine alerts from different sources. The algorithm is presented as following:

```
if EventID is in the queue
then add the detecting info into the slot.
    if a slot is full
    then output data to next processing level.
    else continue.
else create a new slot.
```

Table 1 indicates one instance of the synchronization queue. In this scenario, event 2001 will soon be sent for further manipulation, so event 2002 will move into the first slot. If a new event comes in, it will be put after event 2003.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Table 1. Synchronization queue

Slot	EventID	subject	Verb	Object
1	2001	abnormal	normal	abnormal
2	2002	normal	normal	waiting...
3	2003	waiting...	normal	waiting...

After synchronization, a knowledge based filter is implemented to reduce overhead information for the next processing level. The algorithm is shown below:

`if` $e \in C$
`then` output e to next level
`else` discard e

Where e is the event, and C expresses the critical event set. For instance, a critical event set can be defined in the form: "$*$, $write$, `c:\Windows\*`", which indicates that any process that tries to write files in the folder "`c:\Windows`" will be considered as a critical event. The basic idea of the filter is to discard any event that is not in the critical event set so that it can minimize the load of next processing level.

3.2 Alert normalization level

The second processing level is alert normalization, which transforms different kinds of alert into one consistent set of scale. The STRIDE/DREAD threat model [7], developed by Michael Howard and David LeBlanc in Microsoft, is applied to achieve this.

STRIDE stands for:

- **S**poofing
- **T**ampering
- **R**epudiation
- **I**nformation disclosure
- **D**enial of service
- **E**levation of privilege

And DREAD is an acronym for:

- **D**amage potential
- **R**eproducibility
- **E**xploitability
- **A**ffected Users
- **D**iscoverability

The first step of threat modeling is to create a model of applications, then categorize threats to each attack target node with STRIDE. After that, we can apply a DREAD category to evaluate the intrusion information.

In order to apply the STRIDE/DREAD model, a Categories Threat Level Function (CL Function) is implemented as shown in Fig. 3, where the input of the CL function is the alert and the output is the normalized score of the threat. Parameters of the function include the STRIDE/DREAD coefficients. Here we utilize a matrix with dimension $m \times n$ as the threat modeling coefficients. Normally, m is equal to 6 and n is equal to 5, according to the STRIDE/DREAD. Therefore up to 30 coefficients are involved. These coefficients can be set up in advance manually. A machine learning method such as the Backpropagation (BP) algorithm can be used to improve the CL function in the future.

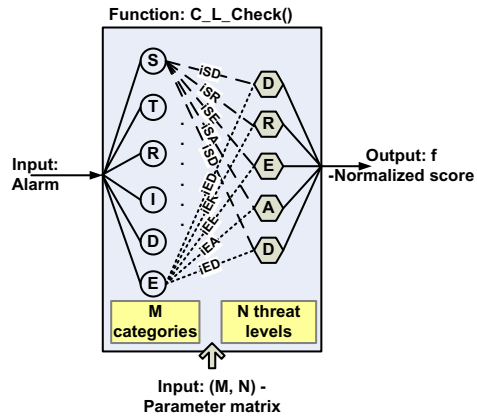


Figure 3. CL Function

3.3 Spacial Alert Fusion level

After the operations of the former processing levels, a threat chart can be obtained. An example is shown in Table 2, where each row indicates threat scores of different features within one event. We called it spacial information of the alert. In this processing level, the "spacial information" as we call it, will be fused into one unique threat information using a two-stage fusing algorithm. The first stage consolidates DREAD scores of different features within SVO models. For example, a common computer process, which is an instance of subject, has three features: user of the process, process identification, and process commonality. User of the process also has three features, which are user location (local or remote), user privilege, and user identification. The DREAD score of the process can be reached using Equation 1.

$$P_x = \sum_{1}^{6} F_x \tag{1}$$

Where P_x means one score within five DREAD categories of the process(subject), and F_x means the corresponding score of one feature. There are six features in our example, so the score of the process is the sum of the six features.

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Table 2. Threat chart

Event	subject	Verb	Object
1	Feature 1 Feature 2 ... Feature n	Operation	Feature 1 Feature 2 ... Feature n
2	Feature 1 Feature 2 ... Feature n	Operation	Feature 1 Feature 2 ... Feature n
...	...	...	...

The second step is to combine the DREAD scores from the SVO monitors into one score of the event. Here a pre-set security index is utilized. For example, a ftp client is more dangerous than Windows NotePAD because it may transfer confidential files. Thus each process is assigned a danger level, which indicates the potential hazard. In the same way, files have their own importance level. So the final DREAD score will be calculated as in Equation 2.

$$E_x = S_x * I_s + O_x * I_o + I_v \tag{2}$$

where E_x is the final score within DREAD categories of the event, S_x is the corresponding score of the Subject, O_x is the DREAD score of the Object, I_s is the security index of Subject, I_o is the security index of the Object, and I_v is the security index of the Verb.

3.4 Temporal Alert Fusion level

Temporal Alert Fusion, the final processing level, is still under development. The basic idea of this processing level is fusing the alerts in a time sequence so that more intrusion information can be acquired. A scenario tree will be built to describe different intrusions. Figure 4 shows a scenario tree [5] [12]. In the figure, the nodes are called scenario nodes,

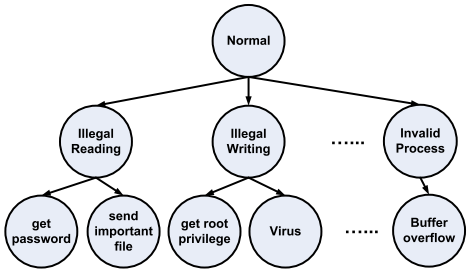


Figure 4. Scenario Tree

which indicate system states. Any path through the nodes is a sequence of intrusions. This processing level only sends output information when the states have changed. Using this tree, redundant alerts can be minimized. For instance, an intruder keeps trying to send a confidential document with high importance level to outside networks. Because

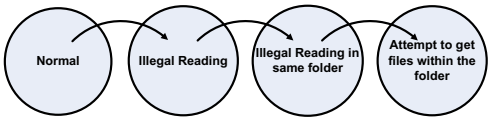


Figure 5. Scenario Path

of Temporal Alert Fusion, the next model of the IDS will receive one alert about this intrusion rather than thousands of duplicated warnings, as shown in Fig 5.

4 DEVS Model Framework

Discrete Event System Specification (DEVS) [14] is applied in this paper to describe the system above. DEVS is a state centered formalism approach which can model systems using an explicit timing specification. According to DEVS system definition, a basic model consists of the following information:

- The set of input ports that receive external events.
- The set of output ports that sent events.
- The set of state variables and parameters.
- The time advanced function which controls the timing of internal transitions.
- The internal transition function which specifies the state transition at the time given by time advanced function.
- The external transition function which specifies how the states changed when an input is received.
- The confluent transition function which decides the transition when conflict happens between internal and external transition functions.
- The output function which gives output before an internal transition.

Events determine values appearing on ports. When external events are received, the model must determine how to respond to them. Meanwhile, the model may change the state automatically. In the following, the DEVS model of our system is discussed in detail.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

4.1 System Design

The DEVS system shown in Fig. 6 is designed according to the IDS system shown in Fig. 1. Borland JBuilder 2005 and DEVS-JAVA 3.0 were used for development. The monitor at the left side is applied to simulate the computer system sensor, which provides the host event to the IDS. Subject checking and object checking are represented as two coupling models of DEVS, which consist of atomic models. Verb checking has been integrated into the alert fusion model for simplicity. Alert fusion and verification models are placed at the right side as atomic models.

The Fig. 7 shows the details of the subject (process) and object (file) checking models. Different feature checking models are represented by DEVS atomic models. The basic model is an abstract model called CheckProcess, which is the parent class of all the feature checkers.

On the beginning, information sent from the lower processing levels will be analyzed by the knowledge based engine. If no data are available, the current fusion cycle is ignored, and nothing is generated. Otherwise, a formalism structure M is used to represent the system:

$$M =< X, S, Y, \delta_{ext}, \delta_{int}, \lambda, ta > \quad (3)$$

where X is the set of input data sources:

$$X = \{(p, e_v)|p \in InPorts, e_v \in Event\},$$

Where $InPorts$ is the set of input ports and $Event$ is the related system event acquired from the system monitor.

Y is the set of outputs:

$$Y = \{(p, e_v)|p \in OutPorts, e_v \in Event\},$$

Where $OutPorts$ is the set of output ports.

$Event = \{SystemInfo\} \times R_0^+$, which is the product of system information and DREAD score. So $e_v = \{(I, DREAD)|I \in \{SystemInfo\}, DREAD \in R_0^+\}$.

S is the set of states:

$$S = \{passive, normal, abnormal\} \times R_0^+,$$

$\delta_{ext}(phase, \sigma, e, X)$ is the external transition function. It will be triggered when some input is available:

`if` $phase = passive$ and $Check() = normal$
`then` $\delta_{ext} = (normal, T_{check}, e_v)$
`else if` $phase = passive$ and $Check()! = normal$
`then` $\delta_{ext} = (abnormal, T_{process} + T_{check}, e'_v)$

Where $T_{process}$ is the processing time of STRIDE/DREAD evaluation; T_{check} is the feature checking time; σ is the time advance, which indicates that after σ time, the state of the system will transit automatically; $e'_v = (e_v.SystemInfo, e_v.DREAD + DREAD)$.

$\delta_{int}(phase, \sigma, S)$ is the internal transition function, which runs when the time elapsed is equal to σ:

$$\delta_{int} = (passive, \infty).$$

$\lambda(phase, \sigma, e_v)$ is the output function, which runs just before the internal transition function δ_{int},

$$\lambda = e_v.$$

Time advanced function ta is defined as:

$$ta(phase, \sigma, (u, v)) = \sigma.$$

The only difference between the various feature checkers is the function $Check()$ within the external transition function δ_{ext}.

4.2 Fusion Model Design

A multi-level alert fusion model is more complex than the checking models. To minimize programming, the first two processing levels, i.e., the Source Preprocessing and Alert Normalization, are put into atomic feature checkers. The function $Check()$ in the external transition function not only provides various feature checking mechanisms, but also does a critical event set check and calculates the related STRIDE/DREAD score. In this paper, the critical set and STRIDE/DREAD matrix are predefined.

Within the alert fusion DEVS model, Spacial Alert Fusion is achieved by equations 1 and 2. Verb checking is also embedded into this model, so that different operations from one process will relate to different security indexes of the subject. The exact index value is set up in advance manually within our simulation.

Various system states and transition links are also defined for temporal alert fusion. Thus the fusion model can eliminate duplicate alerts efficiently.

DEVS model of fusion model is indicated in the following:

$$F =< X, S, Y, \delta_{ext}, \delta_{int}, \lambda, ta > \quad (4)$$

The input set X and output set Y are the same as the ones in feature checking models. In the alert fusion model, there are two input ports, $Inports = \{"Subject", "Object"\}$; and two output ports, $Outports = \{"normal", "alert"\}$.

S is the set of states:

$$S = \{passive, subject, object, normal, alert\} \times R_0^+,$$

$\delta_{ext}(phase, \sigma, e, X)$ is the external transition function, which runs when some input is available:

`if` $phase = passive$ and $X = ("Subject", e_v)$
`then` $\delta_{ext} = (subject, T_{process}, e_v)$

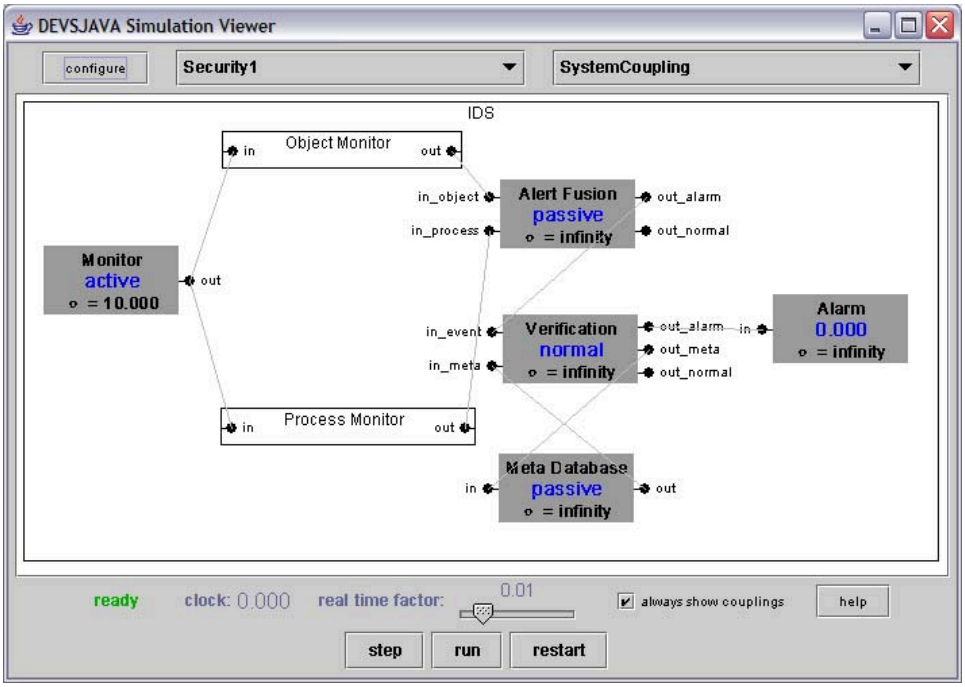


Figure 6. DEVS framework of the IDS

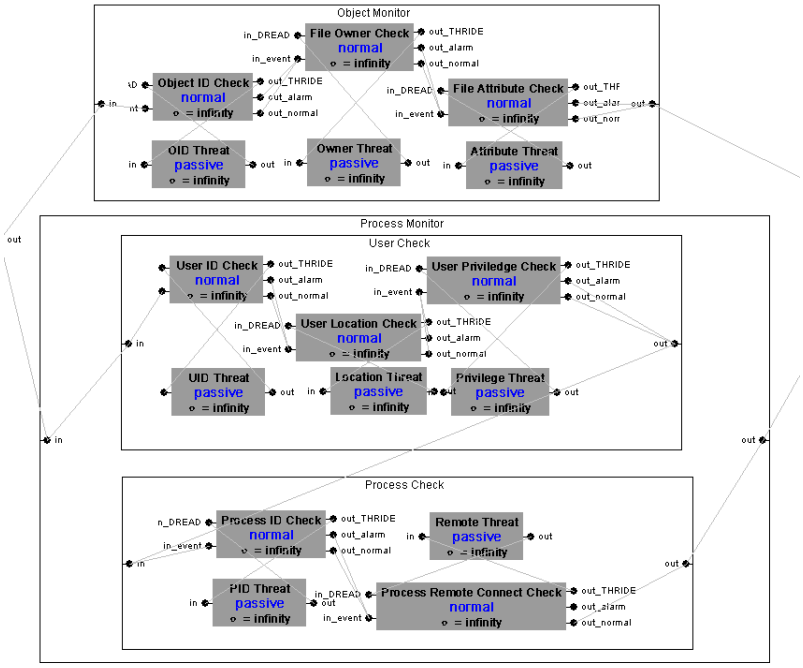


Figure 7. Detail of the coupling models

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

if $phase = passive$ and $X = ("Object", e_v)$
then $\delta_{ext} = (object, T_{process}, e_v)$
if $phase = subject$ and $X = ("Object", e_v)$
then if $Fuse() = false$
then $\delta_{ext} = (normal, T_{process}, e_v)$
else $\delta_{ext} = (alert, T_{process}, e'_v)$
if $phase = object$ and $X = ("Sbject", e_v)$
then if $Fuse() = false$
then $\delta_{ext} = (normal, T_{process}, e_v)$
else $\delta_{ext} = (alert, T_{process}, e'_v)$

$T_{process}$ is the processing time of calculation; σ is the time advance; e'_v is the fused DREAD score after spacial and temporal alert fusion process. The fusion function $Fuse()$ can achieve the calculation.

$\lambda(phase, \sigma, e_v)$ is the output function.

if $phase = normal$
then $\lambda = ("normal", e_v)$
if $phase = alert$
then $\lambda = ("alert", e_v)$

Time advanced function ta is shown as below:

$$ta(phase, \sigma, (u, v)) = \sigma$$

4.3 Simulation

A DEVS-JAVA simulation is described in this section. At first, imitation attack data are utilized to demonstrate the feasibility of the proposed methods. The scenario is shown as below.

First 100 system events are generated. Within the events, 3 intrusions are added. Event number 15 is an intrusion where a legal user tries to access a file which he has no privilege to open. Events number 35 to 45 indicate a scenario where a remote access user tries to access a serious important file that he has no right to access. Event 65 shows an unauthorized user trying to access a file.

The simulation result indicates that 13 alerts have been intercepted, the total number we inserted. The system output is printed as following:

```
Alert! Event ID: 15
Object ID Check
User Location Check
File Attribute Check

Alert! Event ID: 35
User Privilege Check
File Attribute Check
Process Remote Connect Check

Alert! Event ID: 65
User Location Check
User Privilege Check
Process ID Check
```

The result shows the duplicated alerts from event 35 to event 45 have been eliminated due to the temporal alert fusion. The alerts from different features have been integrated together due to spacial alert fusion. Therefore, only three alerts were displayed to the administrator while no useful information was discarded.

5 Conclusion

In this paper, an alert fusion model of the Subject-Verb-Object model based computer host intrusion detection system has been proposed. A multi-level alert fusion model is used to minimize the duplicate alert information from spacial and temporal aspects without losing information. DEVS formalism implementation and DEVS-JAVA simulation is applied to validate the model. Simulation results indicate that the alert fusion system works well.

In the current research, the feature checking threshold and STRIDE/DREAD matrix are predefined. In future research, a machine learning mechanism can be applied to help the system adapt complex intrusions.

References

[1] B. Mukherjee, L. Heberlein and K. Levitt, "Network Intrusion Detection," *IEEE Network*, 1994.

[2] K. Ilgun, "USTAT: A real-time intrusion detection system for UNIX," *Proceedings of the 1993 IEEE Symposium on Security and Privacy*, 1993.

[3] D. L. Hall and J. Llinas, "An Introduction to Multisensor Data Fusion," *Procedings of the IEEE*, Vol. 85, No. 1, pp. 6-23, 1997.

[4] S. Eyles and R. Westgarth, "The Specification of a Submarine Data Fusion System,"*IEEE Colloquium on Principles and Applications of Data Fusion*, pp. 1-8, Fed 4 1991.

[5] S. Mathew, C. Shah and S. Upadhyaya, "An Alert Fusion Framework for Situation Awareness of Coordinated Multistage Attacks," *Processing of the Third IEEE International Workshop on Information Assurance*, 2005.

[6] A. Chan, W. Ng and et al., "Comparision of Different Fusion Approaches For Network Intrusion Detection Using Ensemble of RBFNN," *Proceeding of the Fourth International Conference on Machines Learning and Cybernetics*, pp. 3846-3851, 2005.

[7] M. Howard and D. LeBlanc, "Writing Secure Code," *Microsoft Press*, 2nd edition (December 4, 2002)

[8] N. Ye, J. Giordano and et al., "Information Techniques for Network Intrusion Detection," *IEEE Information Technology Conference, 1998*, pp 117-120, 1998.

[9] A. Siraj, R.B. Vaughn and S.M. Bridges, "Intrusion Sensor Data Fusion in an Intelligent Intrusion Detection System Architecture," *Proceddings of the 37th Hawaii International Conference on System Sciences*, pp 1-10, 2004.

[10] M. Shankar, N. Rao and S. Batsell, "Fusion Intrusion Data For Detection and Containment," *IEEE Military Communications Conference, 2003*, pp 741-746, 2003.

[11] Y. Wang, H, Yang and et al., "Distributed Intrusion Detection System Based on Data Fusion Method," *Proceedings of the 5th world Congres on Intelligent Control and Automation*, pp 4331-4334, 2004.

[12] A. Siraj and R.B. Vaughn, "Multi-Level Alert Clustering for Intrusion Detection Sensor Data," *2005 Annual Meeting of the North American Fuzzy Information Processing Society*, pp. 748-753, 2005.

[13] N. Ye and M. Xu, "Information Fusion for Intrusion Detection," *Proceedings of the Third International Conference on Information Fusion*, pp 17-20, 2000.

[14] B.P. Zeigler and H.S. Sarjoughian, "Introduction to DEVS Modeling and Simulation with JAVA: Developing Component-Based Simulation Models," *Draft Version 3*, 2005

Behavior Analysis-Based Learning Framework for Host Level Intrusion Detection

Haiyan Qiao, Jianfeng Peng, Chuan Feng, Jerzy W. Rozenblit
Electrical and Computer Engineering Department
University of Arizona
Tucson, Arizona, USA
{haiyanq, jpeng, fengc, jr}@ece.arizona.edu

Abstract

Machine learning has great utility within the context of network intrusion detection systems. In this paper, a behavior analysis-based learning framework for host level network intrusion detection is proposed, consisting of two parts, anomaly detection and alert verification. The anomaly detection module processes unlabeled data using a clustering algorithm to detect abnormal behaviors. The alert verification module adopts a novel rule learning based mechanism which analyzes the change of system behavior caused by an intrusion to determine whether an attack succeeded and therefore lower the number of false alarms. In this framework, the host behavior is not represented by a single user or program activity; instead, it is represented by a set of factors, called behavior set, so that the host behavior can be described more accurately and completely.

1. Introduction

With the growing number of network attacks, intrusion detection systems (IDSs) are becoming an integral part of any complete security package of a modern network system. The IDSs perform surveillance and security monitoring of the network infrastructure. A number of different intrusion detection systems have been developed for particular domains (e.g., hosts or networks), in specific environments (e.g., Windows NT or Solaris), and at different levels of abstractions (e.g., kernel-level tools or application level tools). However, computer systems and networks still suffer from an increased threat of intrusions. The existing IDSs are far from perfect and may generate false positive and non-relevant positive alerts [1].

The most widely deployed and commercially available methods for intrusion detection employ signature-based technique, where the signatures or patterns of well-known attacks are provided by human experts. The system or network traffic is scanned for attacks using well-known vulnerabilities and any instances that match the signatures are detected as intrusions. The advantage of this method is a low rate of false positives. The disadvantage is that the signature database has to be revised manually and the system is vulnerable to new types of attack until the revision is done. This limitation leads to active research on intrusion detection techniques based on data mining.

Data mining based network intrusion detection techniques are generally classified into two categories: misuse detection and anomaly detection [2]. In misuse detection, each instance in the training data set is labeled as either normal or intrusion. A machine learning algorithm is trained over the labeled data and then the trained model is applied to classify new data. Misuse intrusion detection is fast, requires little state information, and has a low false-positive rate. With different input data including new types of attacks, the intrusion detection modules are retrained automatically without the manual intervention. However, misuse detection cannot detect novel, previously unseen attacks. Anomaly detection, on the other hand, builds models of normal data to measure a "baseline" of such stats as CPU utilization, disk activity, user logins, file activity, and so on. When there is a deviation from this baseline, an alert is triggered. Currently, almost all commercial intrusion detection systems use misuse detection techniques. Yet, anomaly detection is getting more attention because of its capability to detect novel or unforeseen attacks. Essentially, anomaly detection is the machine learning problem of modeling a normal network or system behavior. Although anomaly detection is becoming an active research topic, widespread adoption of this method faces numerous obstacles, including complexity and high false positive rate.

In order to reduce false and irrelevant alerts, alert verification has to be a part of IDS. Alert verification is

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

a process to determine whether an attack has been successful or not. This information is passed to IDS to help differentiate the type of alerts [16]: 1) The sensor has correctly identified a successful attack; 2) The sensor has correctly identified an attack but the attack failed to meet its objectives; and 3) The sensor incorrectly identified an event as an attack. Alert verification effectively lowers the number of false alarms that an administrator or the decision support system has to deal with.

In this paper, a behavior analysis-based learning framework for host based intrusion detection is proposed, which includes anomaly detection and alert verification. The framework has two characteristics. First, it is learning-based. In the anomaly detection module, a cluster-based outlier detection algorithm detects anomalous data. In the alert verification module, a rule-learning algorithm is applied to learn the behavior changes of the targeted machine. The rules developed serve as an index of alert verification. Second, the framework is behavior analysis oriented. Instead of using a single activity as the indicator of the host behavior, a set of indicators, also called "behavior set", is defined and applied to describe the host behavior. The intrusions are detected and verified based on the analysis of the behavior set. Compared to other research work in host-based anomaly detection, this method does not need high-dimension data since the host baseline is represented by a refined behavior set and each element in the behavior set is of low dimensionality. Thus, state-of-the-art data normalization is not required when outlier detection is applied to detect anomalous data. In section 2, host based anomaly detection is reviewed. In section 3, the behavior analysis-based learning framework is presented and discussed. In section 4, the experimental results are given. Finally, conclusions and future work are summarized in Section 5.

2. Related Work

In host based anomaly detection research, various features are used to model the system behavior baseline, e.g., keystroke characteristics, user command data, system call sequences, file activities, etc. Generally these features fall into three categories: user profile, program profile, and system resource access.

Denning [3] first attempted to build anomaly detection by comparing previous user profiles to current user activity. Sequeira and Zaki [4] designed and implemented a user-profile dependent and temporal sequence clustering-based intrusion detection system by collecting and processing UNIX shell command data. Although analysis of user activity is a natural approach to detect intrusions, experience shows that it is far from accurate. This is because user behavior typically lacks strict patterns.

User dynamics allowed more reflection on the features that define host behavior. All actions carried out by users involve using programs. Programs obtain the required services by executing the specific system call that provides the needed function. Since the code of a given application should not change, the sequence of system calls executed by a program should be regular and predictive.

Most existing research on anomaly detection uses system calls to model system behavior. A number of approaches based on system calls are proposed. Forrest et al. [5] established an analogy between the human immune system and intrusion detection. Lee et al [6] applied a rule learning program to study a sample of system call data. Wagner and Dean [7] proposed to statically generate a non-deterministic finite automaton (NDFA) from the global control-flow graph of the program and simulated NDFA on observed system call trace. Ghosh et al. [10] utilized return address information extracted from the call stack to generate the execution path of a program for anomaly detection. Liao and Vemuri [9] used the k-Nearest Neighbor classifier to classify program behavior represented by frequencies of system calls instead of system call sequence. Eskin et al. [11] applied outlier detection algorithms to anomaly detection using system call data, where the system call data has to be mapped into feature space and the choice of feature space is application specific.

Because the system-call level data is fine grained, it increases overhead and decreases system performance. Thus, some researchers study anomaly detection by modeling files activities. Stolfo et al. [13] studied anomaly detection by learning file system access patterns. Apa et al. [12] noticed that in Windows OS almost all system activities interact with the registry, so they analyzed anomaly detection through modeling normal registry access.

In reality, an attack is usually unpredictable, and it is difficult to know which aspect of the system behavior is associated with the attack. For this reason, modeling system behavior based on only a single category is unreliable, no matter whether the category is user profile, program profile, or system resource access. In what follows, we try to model behavior from all three categories to increase the reliability of system behavior modeling, and thus to increase the accuracy of anomaly detection.

3. Behavior Analysist-Basd Learning Framework

A system consists of both the user and the host machine. It is appropriate to describe the system behaviors using a set of factors of user profile, program

profile, and system resource usage. We call the set of factors the "behavior set."

In this section, a learning framework of intrusion detection is proposed based on analysis of the behavior set, as shown in Fig. 1. This framework includes three modules: anomaly detection, alert fusion, and alert verification. The input of the framework is an event, modeled as a triple $\{subject \times verb \times object\}$ for user/program behavior, and the output is alerts with features of alert ID, alert type, timestamp, priority level, confidence factor, verification status. In the anomaly detection module, the normal behavior baseline is modeled adaptively and stored in the database. When a new event comes, it is classified as normal or intrusion by the module. If an alert is triggered, the alert is fused with other existing alerts to decrease the number of alerts with the same cause. Then the fused alerts are sent to the alert verification module to exclude false or unrelated alerts. In this paper, we discuss only the learning related modules: anomaly detection and alert verification.

To model normal system behavior, both supervised and unsupervised learning algorithms can be applied. We choose unsupervised learning over supervised learning. The main reason is that unsupervised learning does not need labeled (normal/abnormal) data, which is not readily available in reality. Labeled data is generally obtained by simulation or experiments. If labeled data is obtained by simulated intrusions, we are limited to the set of known attacks that we simulated. New types of attacks are not reflected in the training data set. If labeled data is obtained by experiments, then we must face the difficulties in manually classifying large volume of audit data. In addition, if the experimental data labeled normal have buried intrusions, then future instances of those intrusions will not be detected because they are assumed normal in the training set.

Unsupervised learning algorithms take as input a set of unlabeled data and attempt to find noise and intrusions buried within the data. If anomalies are rare, unsupervised learning can be treated as a variant of the outlier detection problem. Outlier based anomaly detections cluster the data based on certain metrics and the data located in sparse regions are claimed as intrusions. Not all intrusions can be detected using outlier based anomaly detection. For example, syn-flood DOS cannot be detected using outlier detection since they are not rare in data distribution. Only when system behavior deviates significantly from average behavior are outlier detections applicable.

3.1. *Outlier detection algorithm*

The outlier detection algorithm we propose is given in Table I. The algorithm is an improvement of fixed-width cluster estimation [11]. For each point, the algorithm approximates the density of points near the given point. The algorithm makes this approximation by counting the number of points that are within a sphere of radius w around the point. Points that are in a dense region of the feature space and contain many points within the circle or ball are considered normal. Points that are in a sparse region of the feature space and contain few points within the circle or ball are considered anomalies.

In the original fixed-width cluster estimation, data that does not belong to any existing clusters is assigned to a new cluster and works as the central point. So the central point of a cluster is sensitive to the sequence of data to be clustered and new data might not be assigned to the closest cluster. In the improved fixed width clustering algorithm, the central point of a cluster is adjusted as new data is added so data are always assigned to the closest cluster. The idea of updating the central point of a cluster is close to K-means clustering, one of the most popular statistical clustering algorithms. Unlike K-means clustering, the improved fixed-width cluster estimation specifies cluster width instead of fixing the number of clusters a priori. Thus the results are sensitive to the value of cluster width. However, this problem can be easily solved by interaction with the clustering results through GUI. Since the clustering is performed offline, it is easy to adjust the width adaptively based on the visual clustering results when the clustering data is of low dimension.

Using outlier detection, it is preferable that the training data be of low dimensionality for the following reasons. First, when distances are measured in all dimensions, it is more difficult to detect outliers effectively because of the average behavior of the noisy and irrelevant dimensions. Second, it is not easy to intuitively explain and understand the clustering results with high dimensions. Third, in high dimensional space, the data is sparse and the notion of proximity fails to retain its meaningfulness. Finally, with increasing dimensionality, it becomes increasingly difficult and inaccurate to estimate the multidimensional distribution of the data points.

Actually, in the learning framework we propose, we do not need to use high dimensionality to represent the feature space of the system behavior. Because the system behavior is described by a set of factors instead of a single complex indicator, all the elements in the behavior set are ensured to be represented by low dimensional data, e.g., the login time, access frequencies of the files, etc.

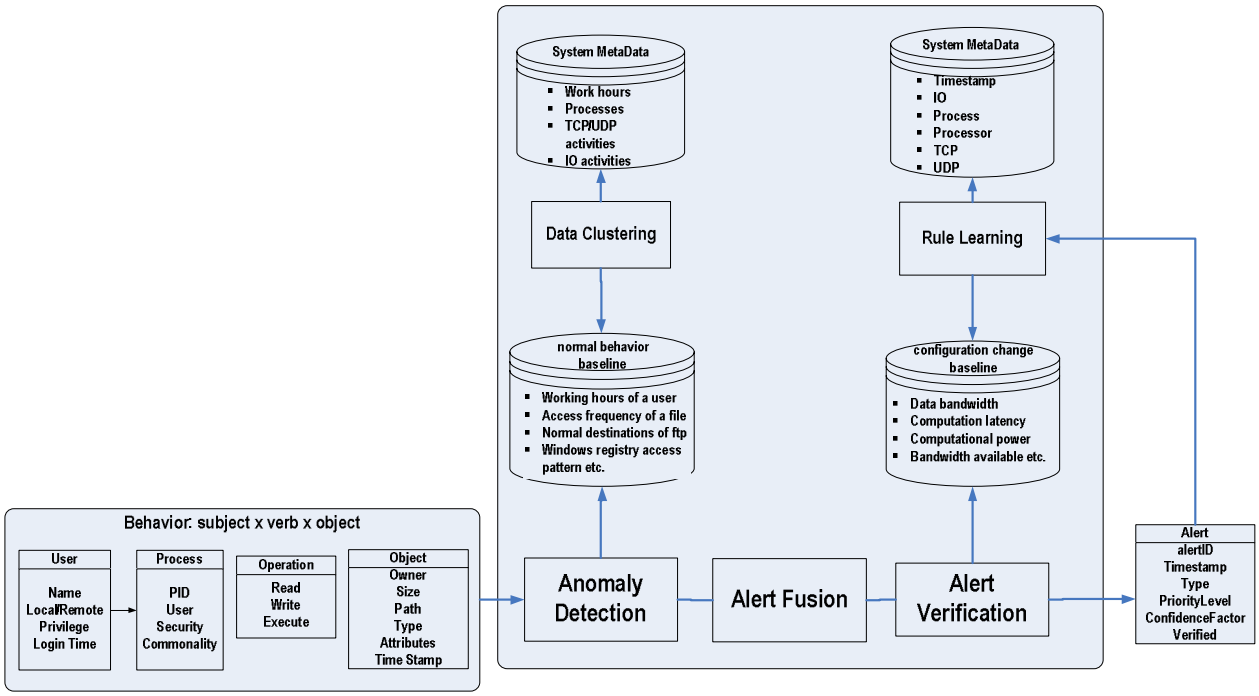


Figure 1. Behavior analysis-based learning framework for host-based intrusion detection

3.2. *Behavior analysis-based intrusion detection*

Anomaly detection can only detect intrusions that make the host behave differently from normal. Thus, before we attempt to model normal behavior, the key question is how to define host behavior. As analyzed at the beginning of the section, we use behavior set to describe system behavior in order to accurately and completely reflect the system characteristics. The behavior set is specified as a set {*user login time, frequency of applications launched, I/O activities, frequency of files access, network activities stats, CPU usage pattern over time, memory usage pattern, data bandwidth, network connection speed*}. In the set, user login time helps to locate normal login time intervals; frequency of application launched locates the most and the least used applications; frequency of files can locate the most and the least accessed files; I/O activities indicates the average bytes written to the files; network activities stats locate normal sequence, frequency, time interval, and other indexes of various network activities, e.g., web browsing, ftp etc.; CPU and memory usages etc. are related to system performance.

To describe the host behavior, we need metadata. A relational database is constructed to store system profiling metadata. The database constructed with WAMP is shown in Fig. 2. In the database, the table features are well designed to reveal the information defined in the behavior set. For example, the table *USER* has features of *userID, user name, login time, remote/local login, logoff time*. The table *PROCESS* has features of *process Id, process name, owner of the process, memory usage of the process, CPU usage of the process, time*.

To maintain data in real time, the data is updated periodically. The time interval to collect data can be modified by the user via a sliding bar in the GUI as shown in Fig. 2. In addition, the user can easily select a table and look at the data through the database interface. When the record size overflows, the new record overwrites the oldest record.

With the profiling database, the outlier detection algorithm is applied to each single factor in the behavior set if applicable. For instance, the user login time is trained to get the normal login time intervals, e.g., [8:00 9:00] and [13:00 14:00]. The CPU usage pattern is learned to model the normal distribution pattern, e.g. peak time at [9:00 11:00] and [14:00 16:00].

Typically, only a few indicators of the behavior set are detected as abnormal. So the significance of the host's deviation from normal behavior has to be measured in some way. In this framework, we adopt the idea of the weighted sum. First, the deviation of each behavior indicator is calculated using the outlier detection algorithm. Then we compute the overall deviation from the host baseline model:

$$(w_1 d_1 + w_2 d_2 + \cdots + w_m d_m)/(w_1 + w_2 + \cdots + w_m),$$

Where m is size of behavior set, w_i is predefined weight of indicator i in the behavior set, and dm is binary value 0 or 1, i.e., normal or abnormal. When the total deviation is beyond a threshold, an alert is generated. The values of w_i $(i = 1,2,\cdots m)$ can be customized based on the host function and the user type. For example, a software developer and an administrator will have very different file activities. If we know the user of the host is a software

developer and the machine is mainly used for software development, we can put more weight on deviations of frequency of applications launched, I/O activities, and frequency of file access.

TABLE I. ANOMALY DETECITON ALGORITHM

Inputs:

Cluster width w

The unlabeled data set D with n dimension

Initialization:

Set of clusters S is empty

Loop until D is empty

For each data d in data set D:

If S is empty

create a cluster C with one element d, add C the cluster to $S\,C \cup S \rightarrow S$, and set d as centroid (x_0, y_0) of C .

Otherwise

find the closest cluster C in S such that for any cluster C' in S, $dist(C, d) \leq dist(C', d)$, where $dist(C, d)$ is Euclidean distance from d to the centroid of cluster C.

If dist(C, d) $\leq w$

insert d into cluster C. adjust the old centroid (x_0, y_0) to $(x_0^{'}, y_0^{'})$, where $x_0^{'} = (m \cdot x_0 + m)/(m+1)$, $y_0^{'} = (m \cdot y_0 + m)/(m+1)$, $m = |C|$ -1.

Otherwise

a new cluster C_d with d as centroid is created, $S \cup \{C_d\} \rightarrow S$.

$D - \{d\} \rightarrow D$

Find outliers:

Sort the clusters based on sizes in ascending sequence, $|C_i| \leq |C_2| \cdots \leq |C_l|$, where l is the number of clusters.

Let $|C_i| + |C_2| + \cdots + |C_l| = p$.

For $i = 1$ to l

If $|C_i| < (p/l) * 20\%$, set the cluster $|C_i|$ as abnormal, i++

Otherwise, set any cluster $|C_j|, j \geq i$ as normal, break

When a new service is installed or the user's behavior changes suddenly, false alerts are triggered. In this case, the fuse module will fuse alerts generated for the same reason. When new data instances are treated as intrusion by the anomaly detection module at the beginning of a pattern change, the new data are not discarded. As the data forming new pattern increases, the host profiling database is updated, and the clustering algorithm is retrained over the new set of data for modeling the emerging new pattern

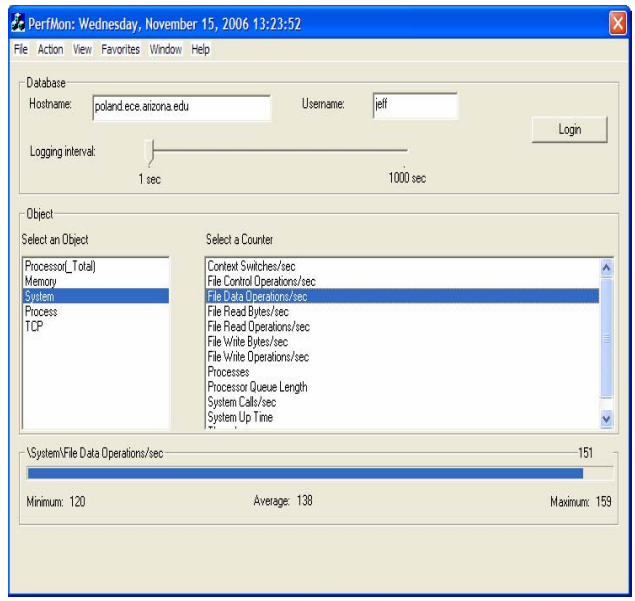


Figure 2. System performance database

When the behavior set is modified, the learning algorithm remains the same, but the database has to be modified or reconstructed.

3.3. *Behavior analysis-based alert verification*

Conventional alert verification approaches include active and passive mechanisms [1]. Passive alert verification mechanisms compare the configuration of the victim machine to the requirements of a successful attack, e.g., a vulnerable version of the Microsoft IIS server. It gathers configuration data before an attack occurs and determines whether the target machine is vulnerable to the attack. Active mechanisms model the expected "outcome" of attacks, checking the visible and checkable traces that a certain attack leaves at a host, e.g., a temporary file or an outgoing network connection. They gather configuration data or forensic traces after an alert occurs. Neither approach can detect unknown attacks. They assume that either the system vulnerability required by an attack or the outcome of an attack is known in advance. If this is true, the attack can be prevented by installing a new service pack. Then there is no need to detect the attack.

In the proposed learning framework, we focus on the verification of unknown and new types of attacks based on host behavior analysis. The after-attack performance measured in quantity might be different in different systems. However, the variance between prior-attack behavior and post-attack behavior is consistent on various machines. For example, an attack causes the network connection to be 50% slower no matter what type of CPU and bandwidth the machine has. Therefore, the basic idea of the alert

verification module is to learn the behavior features associated with an attack and check the change of the value of those features. It is applicable to the attacks that cause system behavior change.

In the training phase, once an alert is verified manually by system administration, the system behavior both before and after the attack occurs are sampled from the system performance database and stored in a separate system metadata database. Since each alert has a timestamp *t*, we just need to probe the system performance database and get the records at time *t* and the last sample time *t-1*. For each system performance-related feature in the records, we calculate the change of feature values:

$$\Delta feature_i = (feature_i(t) - feature_i(t-1) / feature_i(t-1).$$

The system performance features include CPU usage, memory usage, I/O usage, network stats etc. A record in the form of

$$\{alertID, alertType, \Delta feature1, \Delta feature2, \cdots, \Delta feature_l\}$$

is added to the configuration change baseline database, where l is the number of total features, $\Delta feature_i$ is the relative value change of feature *i*. When the configuration change baseline database is constructed, the rule learning algorithm RIPPER [14] will be applied to the database. When applying RIPPER, we get rules for different types of alerts separately. To extract rules for alert type *i* using RIPPER, all the records with that alert type are treated as positive data, all the other records are treated as negative data. RIPPER takes the positive and negative data sets as input, and outputs a rule set represented as a conjunction of conditions in the form of $A_n = v$, $A_c \le \theta$, or $A_c \ge \theta$, where A_n is a nominal attribute and v is a legal value for A_n, and A_c is a continuous variable and θ is some value for A_c.

4. Experimental Results

There are some benchmark data available for network intrusion detection research. For host-based intrusion detection, one data set is from the 1999 DARPA intrusion detection evaluation data which consists of BSM (Basic Security Module) data of all processes run on Solaris machines. Another set of data, obtained from Stephanie Forrest's group at the University of New Mexico [15], contains normal traces for certain programs as well as intrusion traces of system calls for several processes. In our study, these benchmark data are not applicable. First, because we use a novel concept, "behavior set," to describe system behavior instead of using system call traces, and second, because all behavior elements are of low dimension data for the accuracy of clustering. System call data cannot be used for clustering directly and the preprocessing of data is expensive. Eskin et al. [11] adopted spectrum kernels to map the system call data into feature space. However, the feature space corresponding to system calls is in large dimension, e.g., 26 possible system calls and sub-sequences of length 4 will give a dimension of the feature space 26^4, close to 500,000.

To implement the learning framework proposed, we started with system metadata collection. Based on the behavior set we defined, as described in section 3, we constructed the WAMP database server and wrote C++ code to collect the system performance data from a host and store the data into the database. The data will be used as a training set to model the normal behavior of the system. The data is collected in real time and the database is updated dynamically. The data sampling rate is predefined and can be modified through a sliding bar on the GUI as shown in Fig. 2. Fig. 3 shows one of the tables in the system profiling database -- CPU usage data collected from the host.

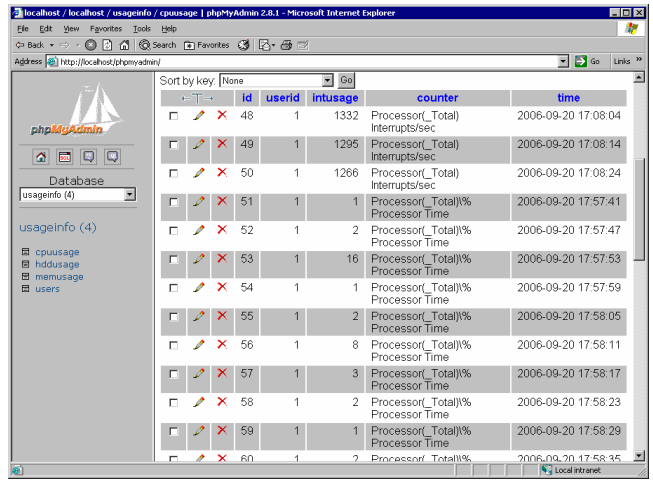


Figure 3. CPU usage data collected from host

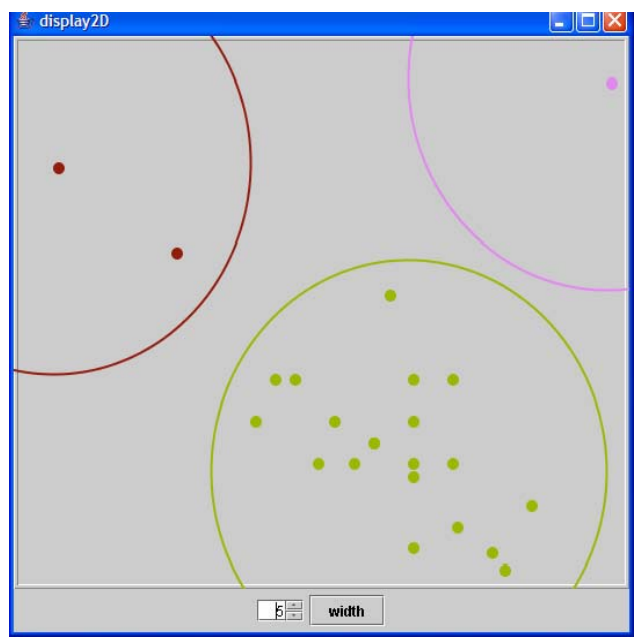


Figure 4. Experimental result of clustering

The unsupervised anomaly detection algorithm is implemented in Java. Given the training data set and cluster

width, the cluster-based outlier detection algorithm is applied and the simulation result is shown in Fig. 4. Using this algorithm, data instances in sparse regions are treated as noise and intrusions buried in the training data, and data instances in dense regions are treated as normal data. When a new data instance comes, whether it is labeled as normal or intrusion depends on which region it is located in. The algorithm needs to specify cluster width instead of the number of clusters. Because the clustering is performed offline, the cluster width can be adjusted easily through the GUI for accurate results.

The connection with the database server is implemented with Java JDBC and the anomaly detection algorithm is also applied to real data from the database. The initial experimental results on a few factors in the behavior set, e.g., the user login time and CPU usage distribution over time, are satisfactory. Further experiments need to be carried out.

To adapt to system behavior pattern changes, the database is updated periodically and the algorithm is applied over the new input data to update the clusters that represent normal behavior.

5. Conclusion

In this paper, behavior analysis-based intrusion detection at the host level has been discussed and a learning frame has been proposed. Two parts in the frame, anomaly detection and alert verification, have been designed using machine learning techniques. The behavior analysis-based framework has the following advantages: 1) the host behavior is described more accurately and comprehensively with a set of indicators, i.e., user profile, program profile, and system resource access; 2) the anomaly detection module does not need labeled data, which are difficult to obtain; 3) each indicator in the behavior set is represented by the data in low dimensionality since the host behavior is refined into a set of indicators; 4) with these low dimension data, the clustering algorithm can be applied over the data without normalization, which is data-dependent and application specific; 5) the clustering algorithm does not need to pre-define the number of clusters, and the width of clusters can be adjusted visually offline; and 6) a novel alert verification approach can check the changes in the host behavior caused by an attack and learn rules associated with the attack.

Currently, the anomaly detection module has been simulated and the host profiling database has been constructed. In the future, the anomaly detection module will be tested on data from a real-world database and the testing results will be carefully examined. In addition, we will implement the alert verification mechanism.

References

[1] F. Valeur, G. Vigna, C. Kruegel, R. A. Kemmerer. A comprehensive approach to intrusion detection alert correlation. 2004

[2] S. Axelsson. Intrusion detection systems: A survey and taxonomy. Technical Report 99-15, Department of Computer Engineering, Chalmers University, March 2000.

[3] D. E. Denning, "An intrusion-detection model." IEEE Transactions on Software Engineering, Vol. SE-13(No. 2):222-232, Feb. 1987.

[4] K. Sequeira, M. Zaki, ADMIT: Anomaly-based Data Mining for Intrusions, Proceedings of the 8th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, Edmonton, July 2002.

[5] S. Forrest, S.A. Hofmeyr, and A. Somayaji. Computer immunology. Communications of the ACM, 40(10):88-96, October 1997.

[6] W. Lee, S. Stolfo, and P.K. Chan. Learning patterns from unix process execution traces for intrusion detection. In Proceedings of AAAI97 Workshop on AI Methods in Fraud and Risk Management, 1997.

[7] D. Wagner and D. Dean, "Intrusion Detection via Static Analysis", *IEEE Symposium on Security and Privacy*, Oakland, CA, 2001

[8] Henry Hanping Feng, Oleg M. Kolesnikov, Prahlad Fogla, Wenke Lee, and Weibo Gong. Anomaly Detection Using Call Stack Information. 2003

[9] Yihua Liao, V. Rao Vemuri. Use of K-Nearest Neighbor classifier for intrusion detection. Computer and Security. Volume 21, Issue 5, 1 October 2002, pages 439-448

[10] A. K. Ghosh, A. Schwartzbard, and M. Schatz. Learning program behavior profiles for intrusion detection. In *Proceedings of the 1st* USENIX Workshop on Intrusion Detection and Network Monitoring. USENIX Association, April 11-12 1999.

[11] Eskin , E, A Arnold, M Prerau, L Portnoy, SJ Stolfo. A geometric framework for unsupervised anomaly detection: detecting intrusions in unlabeled data. In Data Mining for Security Applications, 2002.

[12] Apap, F., A. Honig, S. Hershkop, E. Eskin and S. Stolfo. Detecting Malicious Software by Monitoring Anomalous Windows Registry Accesses. *Fifth International Symposium on Recent Advances in Intrusion Detection*, RAID-2002. Zurich, Switzerland, 2002.

[13] Stolfo, S. J., L. Bui, Shlomo. Hershkop. Unsupervised Anomaly Detection in Computer Security and an Application to File System Access. *Proc. ISMIS*, pp. 14–28, 2005.

[14] Cohen, W. W. Fast effective rule induction. In *Proceedings of the 12th International Conference on Machine Learning.* Lake Tahoe, CA, 1995.

[15] Warrender , C., S. Forrest, and B. Pearlmutter. Detecting intrusions using system calls: alternative data models. In *1999 IEEE Symposium on Security and Privacy*, pages 133–145. IEEE Computer Society, 1999

[16] C. Kruegel and W. Robertson, "Alert Verification: Determining the Success of Intrusion Attempts," *Proc. First Workshop the Detection of Intrusions and Malware and Vulnerability Assessment (DIMVA 2004),* July 2004.

Engineering Vertical Orchestration: from Biometric Trace Events to Incident Reporting

Patricia O'Hagan, Edward Hanna, Roy Sterritt, Paul McKay

Core Systems
Belfast
Northern Ireland
{Patricia, Edward, PaulMc}
@coresystems.biz

University of Ulster
School of Computing and Mathematics,
Jordanstown Campus,
Northern Ireland
R.Sterritt@ulster.ac.uk

Abstract

This paper reports on the latest developments in a deployed Survivable Secure System. The latest version of the system aims to transcribe low-level biometric events into high-level incident reports. This not only provides reactive reporting by providing evidence of who, where, and what when an incident occurs; and pro-active activity by ensuring high-level policies are being followed through low-level system interaction; but also provides another building block towards the long term vision of achieving a predictive system, where through the biometric event trail a developing negative incident can be identified and prevented.

Keywords: business orchestration, autonomic systems, autonomous systems, fault tolerance, biometrics, security, reaction, pro-action, prediction.

1. Introduction

This paper presents a deployed critical security system that requires built in fault-tolerance, survivability and autonomicity as a basis for providing a controlled and safe manageable environment where the participants motives may not always be well intentioned.

This Core System architecture for access control to secure locations such as a correction center incorporates such identification technologies as biometrics. The implication of the security and safety requirements along with these multiple interacting components is that the system becomes more complex, a theme the Autonomic/self-managing systems initiatives aim to address [1]-[5] The continuous monitoring of the system health through control loops matches the architecture needs. Research within the Autonomic field such as self-configuring, self-healing, self-optimizing and self-protection (the *self-chop*) can assist in creating a survivable environment [6]-[9].

The paper expands on previous reports of the system [10] by describing the latest addition to the system that aims to transcribe low-level biometric events into high-level incident reports. This not only has implications of providing functionality for management by providing evidence of who, where, and what; when an incident occurs (reactive reporting) but also assists with pro-active functionality by ensuring high-level policies, as defined by for instance senior management or legislative bodies, are being followed and auditable through low-level system interaction. It also provides another building block towards the long term vision of achieving a predictive system, where through the biometric event trail a developing negative incident can be identified and prevented from occurring.

2. Background and Context

Previously we introduced the deployed Core Survivable Security System (also known as BITS – Biometric Identification and Tracking System) which incorporates biometric technology [10]. In this architecture access to areas and facilities within the secure location is controlled through a distributed survivable system incorporating biometric technology and servers. Due to the nature of these systems – correction centers and secure locations – it is simply not acceptable for the system to go down. As with all systems, components or servers will fail, yet the system must continue to operate in an acceptable fashion that does not put anyone in danger. In the past this fault tolerance would have been achieved through multiple levels of redundancy. Another reality in today's

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

environment is cost – which must be minimized, often removing this option of dual or triple redundancy.

When previously discussing the Survivable Secure System (SSS) [10] we also highlighted the concept of describing the design of a system in terms of high level abstract planes (Figure 1) [11][12].

The Autonomic (quick responding) plane is very much a fault tolerant plane with self-* activity, that is the self-* is motivated to ensure the system is dependable i.e. since system reactions to faults may be quicker than human, or it is composed of operations a human can't deal with it. Yet, in Autonomic Systems, the self-factor is also motivated to reduce the TCO (Total Cost of Ownership) and to deal with or hide complexity, that is, its self-management (through the autonomic and higher level planes) is not just concerned with fault tolerance. The Autonomic research initiatives are also concerned with a top down perspective – setting the business goals or policies and the system manages itself (through self-; configuring, healing, optimizing and protecting) to meet these policies.

This general approach has overtones with policy based management and may be considered vertical business orchestration with transcribing low level activity to high level goals and vice versa through the architecture to meet the company vision, business objectives and critical success factors.

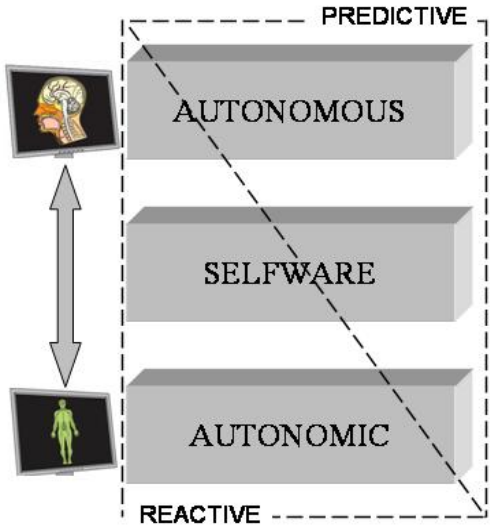


Figure 1 Abstract Planes with Reactive/Predictive Split

3. The Core Architecture

Locations, campuses and sites that require high security access, such as research labs and law enforcement correction centers, are increasingly seeking to utilize the emerging technology in biometrics such as iris and finger print technologies to allow and restrict access and movement around their locations. Secure locations have additional architectural requirements to the norm for dependable systems, such as the architecture should be secure, fault-tolerant, and non-exploitable [14]. That is, the system must meet the security policies of the organization and it must accommodate different security infrastructures; the system must remain robust and secure when faults occur, both random faults caused by the failure of system elements and to malicious faults caused by a deliberate attack on the system; that communication must be reliable so that defensive components remain fully functional even in the face of an attack on the infrastructure; and that it must not be possible for an attacker to exploit defensive components to effect an undesirable action, for example, an attacker or random fault must not be able to trigger a response that causes the system to unnecessarily deny legitimate users access and movement nor the opposite allow non-legitimate users access and movement around the location [14].

As today's systems provide a filler until tomorrow's become a reality, there are many views of a vision to achieve in the long term a highly automated predictive pervasive self-managing environment. Autonomic, pervasive, ubiquitous, ambient, smart-home, service-oriented computation and communications research will be required to meet this vision. Overtime the morphological differences between these research communities will likely diminish as they compete and converge to a general view of future computing and communication systems and environments.

Figure 2 illustrates the deployed Survivable Secure System (SSS) high level module architecture overlaid with approximate classification of the modules into the three architectural abstract planes (as depicted in Figure 1). For more details on these components please refer to [10].

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

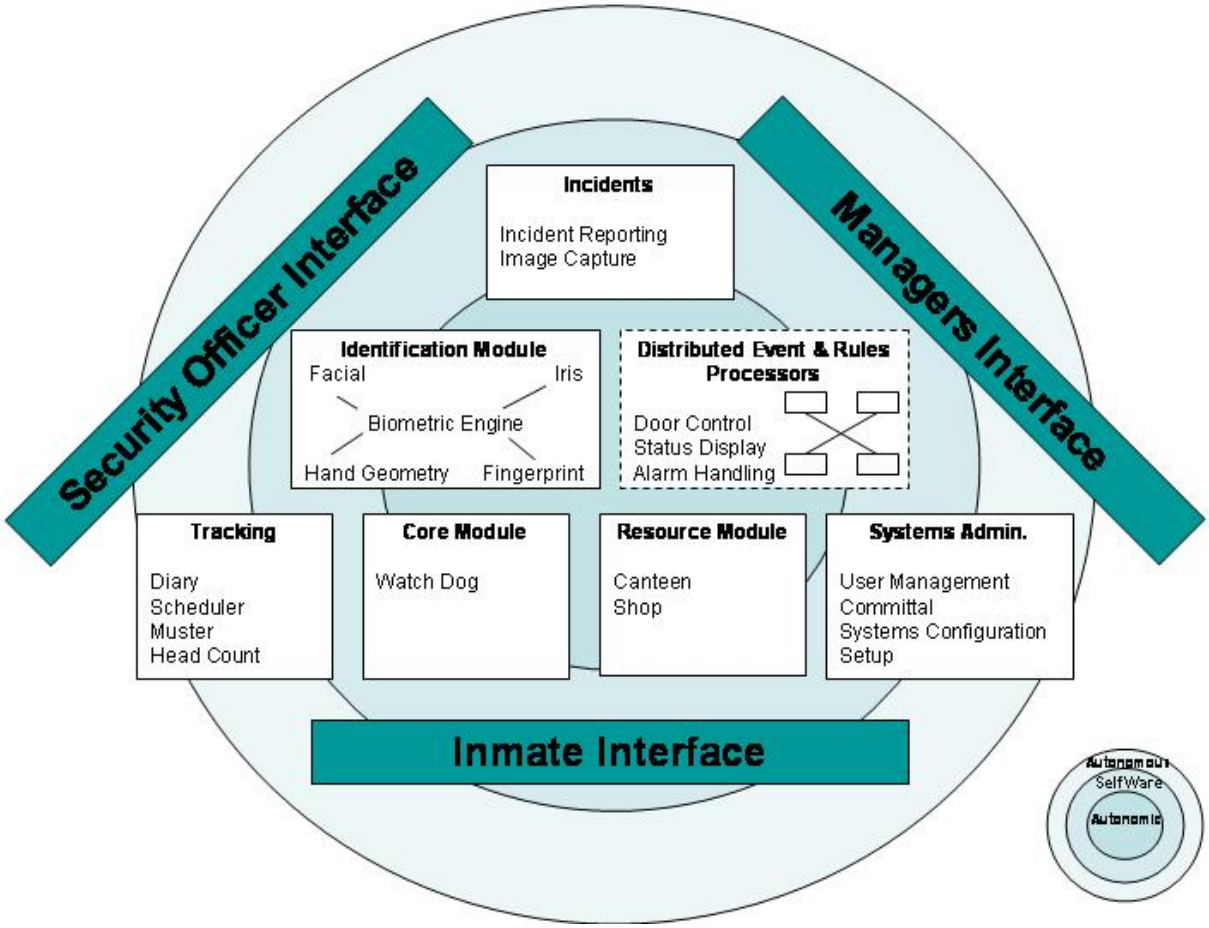


Figure 2 Core Security System - High Level Architecture with Target Self-* Planes

Moving from this current deployed reactive version of the BITS system to the longer term goal of a predictive system had brought about concerns about the high dependence on the ACS (Access Control Server) which processes all events and rules. As such the new version of the system has a distributed event/rules processor to meet the anticipated need of many more events and rules within the system when moving up the planes from the reactive autonomic layer towards, for instance, high level behavioral analysis predicting events in the autonomous layer [12] such as that required for the Incidents reporting system.

The new deployed Incidents Module (IM), which is the first step to enable this vision, is discussed in the following section.

4. Incidents

The new Incidents Module of the BITS software suite provides the user with an investigative tool which can be used to examine the events generated during system operation. This tool can also be used as evidence gathering software, and provides functionality for maintaining a permanent log record of user entries as well as a repository for related files (e.g. documents, images, audio, video).

The Definition of an Incident in this case is

"A significant occurrence or event that interrupts normal system operation"

The GUI of the module provides the user with an interface whereby they can examine and interpret system events. When defining an incident, the user is able to specify a number of parameters which restrict the scope of the incident so that only relevant and pertinent information is included.

The module then displays all of the events contained on the database based on the provided parameters, and provides further filtering and sorting methods to better examine and interpret these.

In effect, the module works both as a retrospective reporting or evidence gathering tool and as a real-time investigative system. There is also the possibility for developing automated responses to system-generated events, such as settings off alarms and enabling a more rapid response to a potentially dangerous situation.

The module also provides the user with a reporting capability to examine past and current incident records on the system. The user can enter a range of parameters and return a dataset of Incidents which can be further sorted, then printed or exported to file.

4.1. System Events

All significant system occurrences are recorded in the BITS database as records in the 'Events' table. Each event record includes a reference to an' Event Type', which provides configurable settings which can determine how the system will respond to the event.

Currently there are approximately 2000 event types defined within the BITS system, and this number is set to grow as further development work is completed. Consequently, the Events table will easily contain several million records after any sustained period of system operation. This made interpretation and examination of the system events almost impossible using the current data structure.

A new data structure has been developed within the Incidents Module to further classify and categorise the existing event types so that it becomes possible to effectively interpret and report on the vast amount of information that is routinely recorded on the BITS system.

4.2. Data Hierarchy

The module provides the user with the ability to create a user definable hierarchical data tree which can be used to categorise the system events as desired. Figure 3 shows an example.

The following data groupings have been defined for use within the module:

- Incident Types: this is the top level grouping, which is used to encapsulate the user-defined event structure. Once defined, the incident type can be selected when starting an incident, and this will associate the events enclosed within the incident type with the newly created incident.
- Event Classes: these groups represent a parent group to which Event Types and child Event Class groups can be added.

These groupings provides a flexible structure which can be user-defined to produce pre-defined incident types at will, which will provide different perspectives on the events stored within the BITS database. The functionality also exists within the GUI to add / remove Event Class groups in real time as the user desires.

In the example, this incident type has been configured so that it will provide an incident which reports on all operational events for the Biometric Server within the BITS module.

The Incidents Module also has its own events, and these are included so that operational events (and errors) which may occur from within the module during the course of the incident are also visible to the user.

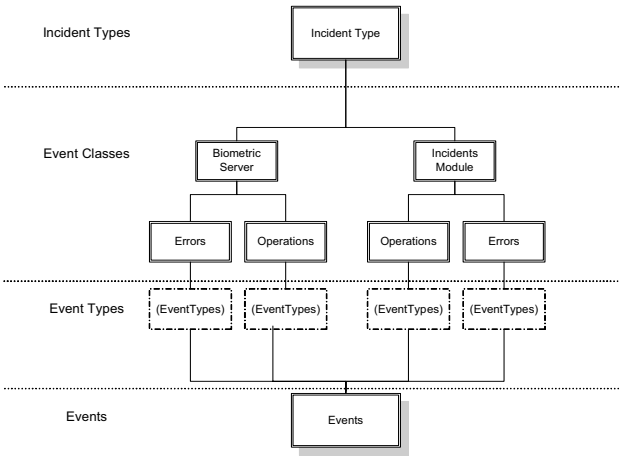


Figure 3 Data Hierarchy for an Incident Type

4.3. Incidents Module Application

The Incidents Module has been presented as a web application using ASP .NET 2.0 and C#. The GUI also makes use of the Microsoft Atlas CTP v2.0.50727 release.

Event classes can be easily configured via the screen displayed in Screenshot 1 below.

This page makes use of the ASP 2.0 TreeView web control, which effectively displays the current data

structure to the user. Event classes can be added and removed at the click of a button.

The right hand side of the page displays the details of the currently selected TreeView node and shows child event classes and all associated event types.

Event Types can be added and removed from event classes via the GUI (Screenshot 2). This screen also incorporates a form for creating a new event type within the BITS database.

In terms of the Incident Module Operation, the main screen from an operational point of view is the Incident Summary screen (Screenshot 2). On this screen, the operator can control every aspect of the Incident.

The following functionality is incorporated in this screen:

- Incident Log:
- Attachments: the module includes a document repository so that the operator can upload relevant files and attach them to the incident so that they can be viewed at a later stage (in read-only format).
- Incident Properties: all parameters relating to the incident can be edited from this page.
- Event Logger (Screenshot 3): this GridView runs using an Atlas Timer Control to continually update and display the latest relevant system events to the user. There is also a filter which allows for addition and removal of Event Classes to further sort the displayed events as the user desires.

4.4. Improvements due to Incidents Module

The direct improvements that the deployed IM have brought to the system are as follows:

1. Visibility - the system generates a high volume of events. It is difficult to see what is happening when something goes wrong. The IM provides the ability to isolate certain events associated with a particular type of incident. Hence the fault finding processes is improved, - easier and faster to isolate contributing factors.

2. Pattern recognition - the IM can be used to gather data on a number of similar incident types - this could be examined for patterns and possibly help prevent further occurrences of the incident.

3. Accountability - provides definitive evidence (against timeline) to support that procedures were followed according to company policy or that this did not happen. Can be used to provide evidence when security breach arises, or cases of alleged negligence.

4. Export facility - the IM enables incident evidence to be exported for external evaluation.

As has been highlighted, the IM represents an innovative evolutionary step on a much longer journey to provide a highly predictive and reactive self-managing secure biometric identification and tracking environment. It has been designed to not only provide functionality to the current stake holders but also assist in engineering the next version of the system.

5. Conclusion

As we move towards future computing and communication paradigms with thousand and potentially millions if not billions of interconnected devices, the need for *selfware* as an intrinsic part of these systems becomes clearer. Some of the issues for these future ubiquitous/pervasive computer-based systems are in effect emerging with respect to the application and system discussed in this paper. The biometric system with pervasive devices such as biometric components, CCTVs, computing and comms infrastructure as used to provide access, identification, tracking, and prevent undesirable incidents is demonstrating the issues of even further complexity in the systems and the need for self-management and survivability throughout all planes of the architecture.

This paper focused on the need for vertical orchestration (depicted through abstract architectural planes: autonomic / selfware / autonomous.) to transcribe low-level biometric events into high-level incident reports. This not only provides reactive reporting by providing evidence of who, where, and what when an incident occurs; and pro-active activity by ensuring high-level policies are being followed through low-level system interaction; but also provides another building block towards the long term vision of achieving a predictive system, where through the biometric event trail a developing negative incident can be identified and prevented.

Acknowledgements

Core Systems' development project is partly supported by InvestNI. The wider context of the Autonomic Systems research is supported at the University of Ulster by the Computer Science Research Institute (CSRI) and the Centre for Software Process Technologies (CSPT), funded by Invest NI through the Centres of Excellence Programme, under the EU Peace II initiative.

References

[1] P. Horn, “Autonomic computing: IBM perspective on the state of information technology,” IBM T.J. Watson Labs, NY, 15th October 2001.

[2] R. Sterritt, "Autonomic Computing", Innovations in Systems and Software Engineering, Vol. 1, No. 1, Springer, pp 79-88, 2005

[3] MG Hinchey, R Sterritt, "Self-Managing Software", Computer, Vol. 39, No. 2, IEEE Computer Society, doi:10.1109/MC.2006.69, Pages 107-109, Feb. 2006.

[4] R. Sterritt, “Towards Autonomic Computing: Effective Event Management”, Proc. IEEE/NASA SEW, Greenbelt, MD, doi:10.1109/SEW.2002.1199448, Dec. 2002.

[5] JO Kephart, DM Chess. “The Vision of Autonomic Computing”, Computer, 36(1):41–52, 2003.

[6] R. Sterritt, D.F. Bantz, “PAC-MEN: Personal Autonomic Computing Monitoring Environments,” Proc IEEE DEXA 2004 Workshops - 2nd Int. Workshop on Self-Adaptive and Autonomic Computing Systems (SAACS 04), Zaragoza, Spain, Aug. 30 – 3 Sept., doi: 10.1109/DEXA.2004.1333562, 2004.

[7] R. Sterritt, M.G. Hinchey, (Apr 2005) "Birds of a Feather Session: “Autonomic Computing: Panacea or Poppycock?”", Proceedings of IEEE Workshop on the Engineering of Autonomic Systems (EASe 2005) at 12th Annual IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS 2005), Greenbelt, MD, USA, 3-8 April, 2005, doi: 10.1109/ECBS.2005.22, Pages 335-341

[8] R Sterritt, DW Bustard, "Autonomic Computing: a Means of Achieving Dependability?", Proc IEEE Int. Conf. on the Engineering of Computer Based Systems (ECBS'03), Huntsville, Alabama, USA, April 7-11 2003, doi: 10.1109/ECBS.2003.1194805, pp 247-251.

[9] R. Sterritt, “Pulse Monitoring: Extending the Health-check for the Autonomic GRID”, Proceedings of IEEE Workshop on Autonomic Computing Principles and Architectures (AUCOPA 2003) at INDIN 2003, Banff, Alberta, Canada, 22-23 August 2003, doi: 10.1109/INDIN.2003.1300375, pp 433-440

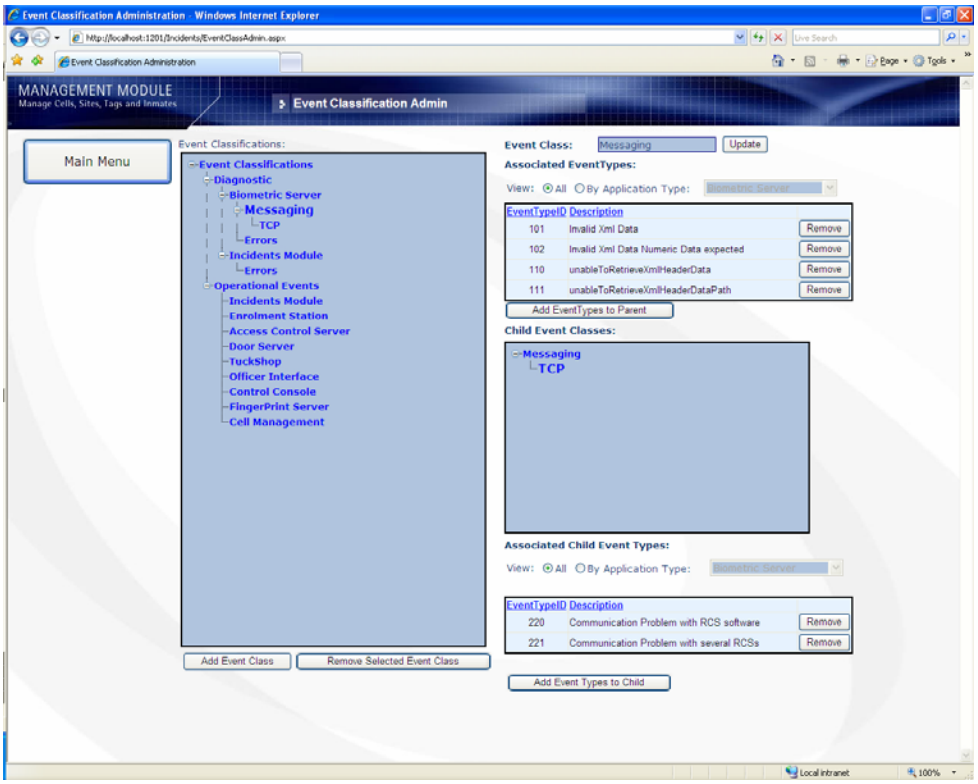
[10] R Sterritt, G Garrity, E Hanna, P OHagan, (Sep 2005) "Survivable Security Systems through Autonomicity", Proceedings of 2nd NASA/IEEE Workshop on Radical Agent Concepts (WRAC 2005), NASA GSFC, Maryland, USA, 20-22 September 2005, in "LNAI", Springer.

[11] R. Sterritt, G. Garrity, E. Hanna, P. O'Hagan, “Autonomic Agents for Survivable Security Systems", 1st IFIP Workshop on Trusted and Autonomic Ubiquitous and Embedded Systems (TAUES 2005) at EUC'05, Nagasaki, Japan, 6-9th December, in "LNCS 3823", Dec. 2005

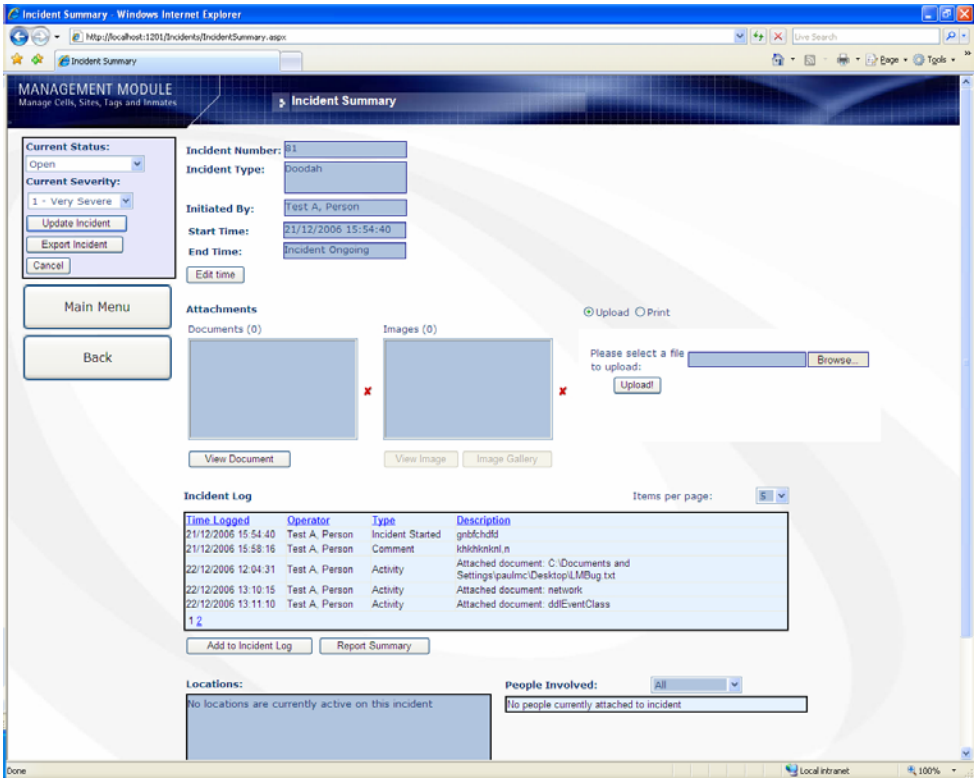
[12] R Sterritt, E Hanna, P OHagan, G Garrity, (Apr 2006) "Targeting Prediction Engineering a Distributed Event Processor for an Autonomic Biometric System", Proceedings of 30th NASA/IEEE Software Engineering Workshop (SEW-30), Columbia, Maryland, USA, April 24-28

[13] Lightstone S, “Foundations of Autonomic Computing Development”, IEEE TFAAS Letters, Mar 2006, http://tab.computer.org/aas/

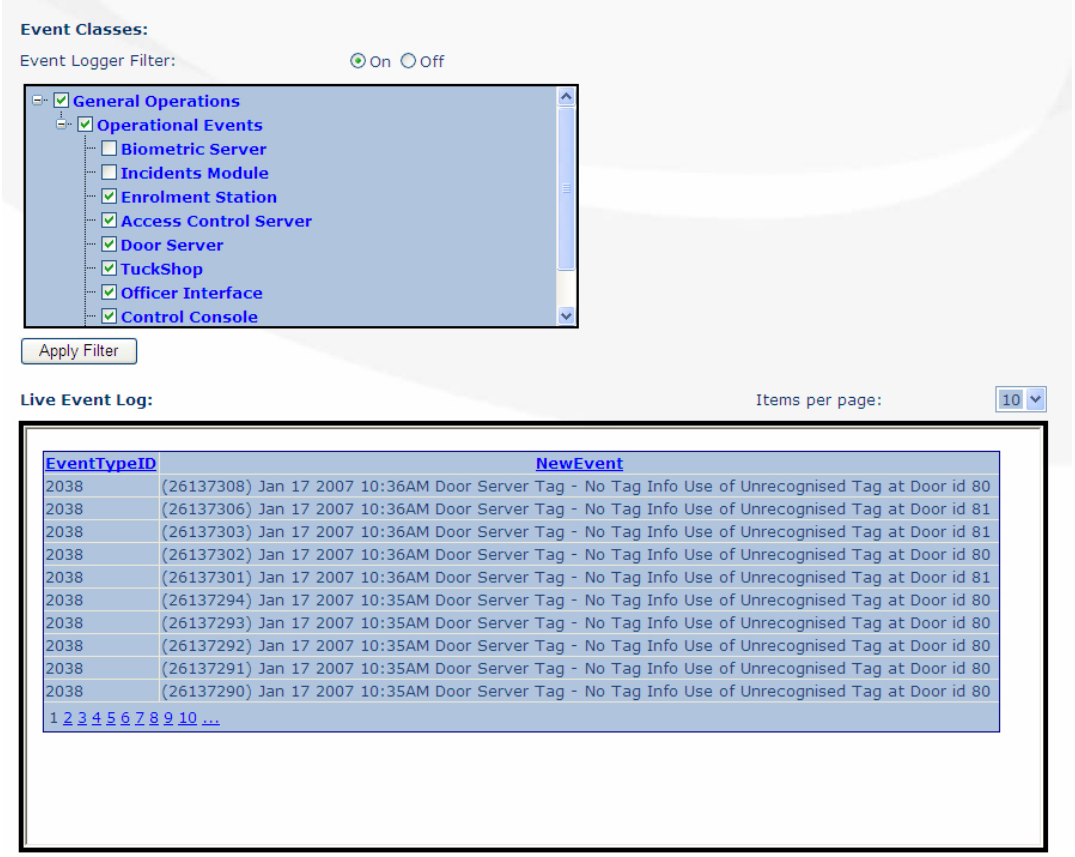
[14] SM Lewandowski, DJ Van Hook, GC O'Leary, JW Haines, LM Rossey, “SARA: Survivable Autonomic Response Architecture”, DARPA Information Survivability Conference and Exposition II Proceedings, Vol. 1, pp. 77-88, June 2001.



Screenshot 1 Event Type Setup



Screenshot 2 Incident Summary Screen



Screenshot 3 Detail of Event Logger and Filter

Validation of Component-based Software with a Customer Centric Domain Level Approach

Oliver Skroch
Business Informatics and Systems Engineering
Universität Augsburg, Universitätsstr. 16, D-86159 Augsburg, Germany
oliver.skroch@wiwi.uni-augsburg.de

Abstract

End user testing for higher-order software compliance becomes an issue of increasing importance with compositional reuse of software artifacts. Few if any existing approaches discuss the validation of higher-order domain aspects in this context. The proposed method derives testable validation scenarios directly from a customer domain model by abstraction, reduction and inclusion for critical coverage. The resulting branch-free validation scenarios are used as references to validate suppliers' software specifications against. Advantages of the method are first the embedding into a clear business model, second test oracles that originate from the business domains and thus are independent from models within the software development process, and third the early availability of validation results in the development cycle, before the software itself is available.

1. Introduction

In 1969 there was already a proposal for the reuse of software parts, with components that can be looked up in catalogues and can then be integrated into large applications similar to electronic parts [15]. Reuse was later described as the only realistic approach to meet the needs of the software industry [18]. Recently, further increasing needs for reuse have been listed among the top trends that will influence future software processes [5].

Compositional reuse is one of the fundamental software reuse technologies [4]. The idea is to reuse executable artifacts from repositories for the composition of larger applications [29]. Compositional reuse of black box business components is part of the overall concept of component-based business applications, where business components are described by multi-layered specifications, implement services from a business domain, and are envisaged to be traded on markets [32].

In such an environment, customers and users of component software do not want to access source code but restrict to a black box view. They focus on "higher-order" [20] fit of domain level pragmatics and semantics, while syntactical compliance of formal and mere technical aspects are often perceived as the suppliers' responsibility.

Non-trivial problems still complicate broad compositional software reuse in theory and in practice. Among the problems is the evaluation of available components against these more complex end user domain requirements by appropriate validation testing.

The importance of testing is well acknowledged in traditional engineering disciplines because of their long history of experience. Software is on the one hand fundamentally less reliable than traditional engineering products [25]. On the other hand the well-known notion of "good-enough software" [37] shows a pragmatic view on quality aspects of software, in particular with large enterprise applications.

But also good enough enterprise software development can profit from testing on the bottom line, especially if errors are found efficiently and early in the development process. Firstly, it was shown that the effort for error rectification grows markedly when the error is detected later. Secondly, the earlier errors are detected the more corrective alternatives are available. Additionally, studies in science and projects in industry indicate that testing takes more than fifty percent of the effort even with non-safety critical software.

Software testing is even more important whenever prefabricated items such as components are reused. Firstly, a single component made for reuse must be more thoroughly tested than a component made to be used once because it is reused in combinations unknown at the time of development. Secondly, a system based on a configuration of multiple black box

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

components from different suppliers must be more thoroughly tested as compared to large pre-integrated products. [17]

The distinction between technology based supplier testing and domain based customer testing is widely acknowledged, in particular with component-based software [35]. The proposed validation approach improves component validation testing on the domain level and thus contributes to advances in software testing and in compositional software reuse.

The rest of the paper is structured as follows. Section two of the paper sets out basic assumptions and presents the underlying business model. Section three presents the research approach: it states the research questions, introduces the proposed solution approach in an overview, and applies the method as proposed by now in a small example, non-fictitious on the domain side. Section four elaborates on the current state of the art and on existing solutions, delimits the contributions of the proposed method, and lines out results achieved and next steps. Section five summarizes this paper.

2. Basic assumptions and business model

The proposed approach is based on the fundamental assumption that the final arbiter of software success is only the customer to whom the component software is useful or not. This most central assumption was stated already in 1979: *"A software error is present when the program does not do what its end user reasonably expects it to do."* [20].

From an end user's domain point of view, it is favorable to validate higher-order requirements independently and as early as possible, to support the identification and assessment of components before the executable software itself is available. The validation knowledge about testable business requirements that predefine what a software solution is supposed to do needs to be constructed. Domain level testing, without the intention to change or reengineer components or their specifications, initially has only one goal: to validate the suppliers' software for reuse and control. This argument of assertive and independent consideration of the ontological domain and the supporting technologies can be founded in Ψ (psi) theory [8].

In the method proposed here, customer test references from the application domain prevail over an oracle created with mere supplier knowledge from within the component software technology. The proposed approach is embedded into a clear business model assumption derived from an overall vision of industrialized compositional reuse for software engineering, which has been described in detail in [32]. Figure 1 (notation: e3-value [11]) introduces the underlying business model assumption with the three actors: component supplier, component customer and component market.

In the underlying concept, suppliers create components for an anonymous market to satisfy an assumed demand or requirement on that market. These requirements can be acquired from discussions with an individual client but are more typically entrepreneurial market assumptions. Offered software components are technically mature and suppliers keep their source code undisclosed. They completely specify their components in black box style by fully defining the interfaces to convey the components' contracts (what the components do) but without disclosing their implementation details (how the components work) [16]. Specifications achieving this are multi-layered and semi-formal today. Respective specification approaches are proposed e.g. in [31] where contract levels and facts to be specified describe the external view onto the component for reuse. These component specifications serve as black box description for reuse and are put into publicly available component specification libraries.

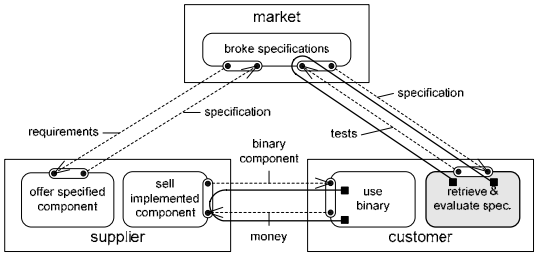


Figure 1. Business model assumption

Component software users want support and automation for their requirements and search a wide variety of library components. The available components are found as specifications e.g. on the Internet. Customers query the black box functional specifications with specific predefined criteria, retrieve matches, and then evaluate the retrieved specifications in detail. Both retrieval and evaluation imply a comparison i.e. a test between reference features demanded and specification candidates offered.

An important challenge for black box reuse at this point is how to derive reasonable specification retrieval and evaluation criteria, and that means: how to test end user domain requirements vs. the supplier specifications.

The associated testing may be classified as specification based or program based, and specification based testing can be divided into state

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

based testing and black box testing [33]. The component paradigm of the described business model assumes that components are tested on the basis of their specifications, and restrict the approach to black box testing. It is acknowledged that good overall testing will be comprehensive and will employ a set of complementary methods in practice.

3. Research Approach

3.1. Research questions

Dynamics and pragmatism of real life businesses demand "good enough" software which is useful to the customer, and therefore support a focus on higher-order domain validation tests: *How are appropriate and independent higher-order domain validation test references created? From what basis can they be derived, how, and how are they documented?*

Suppliers' semi-formal, multi-layered component reuse specifications represent the candidates for early domain validation testing: *Which parts and aspects of a full black box specification can reasonably be validated without actual software being available? How exactly are the suppliers' component specifications compared to the customers' domain validation scenarios?*

Testing alone cannot improve the quality of software, but early and expressive test results can improve decisions: *Which tangible results can domain validation produce on the basis of reuse specifications? What decisive conclusions can be drawn from the results?*

3.2. Proposed solution approach

Customer validation scenarios are used to systematically validate aspects of multi-layered component reuse specifications, if possible showing that the specified software works for the higher-order domain requirements.

To validate requirements they must be stated in testable terms. The starting point is the observation that also for validation of higher-order domain functionality, only a small subset of the full domain is actually relevant for the end users' intended automation with distinct effects on utilized system behavior.

Figure 2 (notation: activity diagram [22]) gives an overview on the proposed ARIval (abstraction, reduction, inclusion, validation) method. To construct testable business requirements on the customer side, the starting point is a domain model which, in many cases, is available through prosaic business rules and process descriptions as semi-formal or informal model, e.g. activity diagram, event driven process chain, Petri net, etc. Full or partial automation is required for the model from ready-made software components.

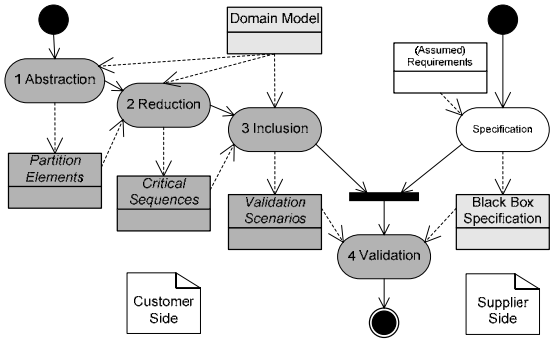


Figure 2. Proposed method overview

Relevant parts of the model environment are first *abstracted* through the well-known equivalence partitioning and boundary value analysis [20]. This results in partition elements which are a categorization of the original domain, with one representative element per partition.

The abstracted elements are then *reduced*, by identifying reasonable and critical sequences. Complexity of typical requirements in real settings will lead to very many possible sequences at this point and prevent an exhaustive validation. This means that with each possible sequence of steps that requires automation on the domain side, and with the corresponding sequence of equivalence classes, a number of validation critical sequences need to be selected from the large number of all possibilities. Reduction criteria are domain centric and come from outside of the software engineering process. They include domain considerations e.g. on frequency, criticality, financial risk, external visibility, etc. instead of software centric objectives such as coverage of all control statements in the code. The abstracted and reduced domain part then contains value representatives in sequences, with each sequence deemed critical by the customer for the intended application.

An *inclusion* will use the critical sequences to build validation scenarios, both within a domain part and across a number of different related domain parts, to cover the validation critical paths in context. These scenarios must not contain branching but make up simple linear paths in order to avoid quantitative evaluation problems during actual validation. To achieve this, a branching critical sequence is iteratively included as two or more branch-free scenarios, until all relating scenarios are linear. In this way each linear

scenario is deliberately and consciously included into the validation step, or not. Inclusion criteria, again, are domain centric and are derived from considerations rooting in domain ontology instead of software technology, as described. The expected positive or negative results are set for each scenario now, too.

Finally, the actual *validation* will numerically check the applicable parts of the reuse specifications using all formally defined and branch-free critical validation scenarios as test cases.

Two basic validation coverage measures can be defined from the abstracted domain, which is an equivalent of the original domain. Reduction coverage measures abstracted domain requirements against reduced sequences. Inclusion coverage measures reduced sequences against included scenarios. Both measures could be plain and weighted. The weighted coverage would scale on numeric scores given for each reduction and inclusion criteria, e.g. by using a simple ranking.

Beneficiaries of the method are mainly customers and end users in the presented business model. The validation method supports them to evaluate the many component specifications from repositories on the basis of their testable requirements, independently derived from their ontological domain, and before actual software is available.

3.3. Example

The example is taken from a large company's business rules and processes for the creation of credit items. Figure 3 shows one function out of the process diagram and the relevant business rules for this "authorization level ok?" process step.

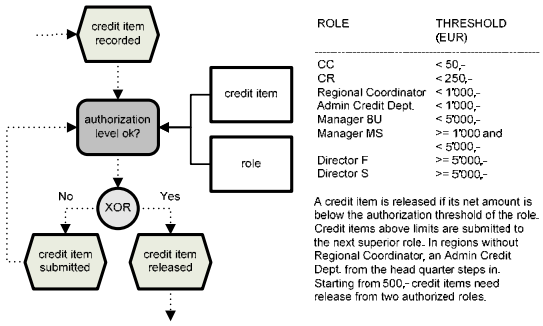


Figure 3. Excerpt from domain model

A credit item has been recorded by an agent at this stage, now it needs formal release. Everyone involved in the process belongs unambiguously to a certain role, and all roles have limits for releasing (rel) a recorded credit note depending on its amount. If the credit amount is above the role's limit, it is not released but instead submitted (sub) to the next superior role. Above a certain amount, any credit needs release by two different authorized roles (rel-s, rel). The two highest roles are entitled to release all credits. In the described process, credits that shall not be released remain in an undefined, or submitted, state.

Abstraction maps the original domain model onto an equivalent domain model with defined partitions and distinct value representatives per partition. The example results in seven partitions shown in Table 1 together with their value representatives. If the analyzed customer domain section does not define any interface behavior, e.g. for partition P_1 in this example, then tests cannot be derived from this part of the domain model.

Partition	Value	Partition	Value
$P_1 =]-\infty, 0]$	$e_1 = -1$	$P_5 = [500, 1000[$	$e_5 = 500$
$P_2 =]0, 50[$	$e_2 = 25$	$P_6 = [1000, 5000[$	$e_6 = 1000$
$P_3 = [50, 250[$	$e_3 = 50$	$P_7 = [5000, \infty[$	$e_7 = 5001$
$P_4 = [250, 500[$	$e_4 = 250$		

Table 1. Partitions and value representatives

Reduction identifies the data sequences that are critical and reasonable for validation from the full set of possible sequences, from an end user validation point of view. The paper restricts to demonstrating positive sequences, negative sequences work according to the same principle. The example can first be reduced from an end user's business perspective to sequences starting at the least empowered call center (cc) role, which will subsequently cover also superior roles with suitable partition values. This reduction results in Table 2 listing ten sequences, the validation critical scenarios through the domain section.

Role	cca	cr	rc	acd	mbu	mms	df	ds
S_1	e_2 rel	-	-	-	-	-	-	-
S_2	e_3 sub	e_3 rel	-	-	-	-	-	-
S_3	e_4 sub	e_4 sub	e_4 rel	-	-	-	-	-
S_4	e_4 sub	e_4 sub	-	e_4 rel	-	-	-	-
S_5	e_5 sub	e_5 sub	e_5 rel-s	-	e_5 rel	-	-	-
S_6	e_5 sub	e_5 sub	-	e_5 rel-s	e_5 rel	-	-	-
S_7	e_6 sub	e_6 sub	e_6 sub	-	e_6 rel-s	e_6 rel	-	-
S_8	e_6 sub	e_6 sub	-	e_6 sub	e_6 rel-s	e_6 rel	-	-
S_9	e_7 sub	e_7 sub	e_7 sub	-	e_7 sub	-	e_7 rel-s	e_7 rel
S_{10}	e_7 sub	e_7 sub	-	e_7 sub	e_7 sub	-	e_7 rel-s	e_7 rel

Table 2. Critical sequences

These sequences are now eligible for inclusion, also with critical sequences from other, related domain parts, to build end-to-end branch-free business validation scenarios. The approach to connect sequences is the same as it was shown for the steps within a domain part. Joining two scenarios becomes possible by using the preceding scenario's output as

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

the subsequent scenario's input. Inclusion criteria, again, are fully domain centric.

To demonstrate the validation of a specification artifact on the basis of the sequences in Table 2, it is assumed that a software provider has specified and offered a `Comparator` software component. Next to other levels and facts, the behavior of this software artifact is described in OCL (Object Constraint Language) [21], and a `checkGE` service ("greater or equal") is defined according to Listing 1. It is also specified for the `Comparator` component, on the respective level of the multi-level reuse specification, that a `limits` relation maps one `value` to one `actor`.

To validate the behavior specified in Listing 1 against customer requirements given as critical sequences, the constraints from the supplier's specification are now numerically checked against one ore more branch-free validation paths. Such a path can be one path through several subsequent critical customer sequences from interrelated domain parts that are assembled and validated together.

This example restricts to demonstrate the case of a single critical sequence, S_4 from Table 2. In natural language, S_4 follows a recorded credit item of 250.- from a region without regional coordinator role. The credit item is (i) beyond the credit authorization limit of the call center role and therefore submitted to the customer representative role. It is (ii) beyond the credit authorization limit of the customer representative role and therefore submitted to the administrator credit department role. It is (iii) within the credit authorization limit of the administrator credit department role and released. Validation of this sequence is done by systematically walking through the OCL constraints from Listing 1.

In step (i) the first two preconditions hold: `val` is `250` and `act` is `cca`. The third precondition also holds: once the mapping table is set up with role descriptions and thresholds from the domain model, then `cca` will be found in the `limits` relation. If the preconditions hold as described, the specification's postcondition will evaluate `(50 >= 250)` and return `false`. The work flow can identify this with the meaning that the credit item is not released, and return to the "authorization level ok?" function with a "credit item submitted" state.

In step (ii) the first two preconditions hold: `val` is `250` and `act` is `cr`. As in the previous step the third precondition also holds for `cr`. If the preconditions hold as described, the specification's postcondition will evaluate `(250 >= 250)` and return `true`. The work flow can identify this with the meaning that the credit item is released, and continue to further parts of the domain model with a "credit item released" state.

In step (iii) the first two preconditions hold: `val` is `250` and `act` is `acd`. As in the previous steps the third precondition also holds for `acd`. If the preconditions hold as described, the specification's postcondition will evaluate `(1000 >= 250)` and return `true`. The work flow can identify this with the meaning that the credit item is released, and continue to further parts of the domain model with a "credit item released" state.

```
context Comparator::checkGE( val:Real, act:String ):Boolean

pre:
( oclIsUndefined( val )=false )
  and
( oclIsUndefined( act )=false )
  and
self.limits->
  exists( actor:String | actor=act )

post:
if self.limits->
     select( actor:String | actor=act ).value >= val
  then result=true   -- greater or equal
  else result=false  -- not (greater or equal)
endif
```

Listing 1. Behavioral specification artifact (OCL)

Thus, on the bottom line, validation of the `Comparator` component vs. sequence S_4 using the ARIval method revealed a problem. While steps (i) and (iii) can be performed correctly by the specified software, in step (ii) the `Comparator` component fails validation vs. the business rules. In the domain model and its critical sequence S_4, the credit item of 250.- is not released by a customer representative but instead submitted to be checked by the superior role. In the `Comparator` component, the validation shows that the credit item of 250.- is actually released by the customer representative role, which is inconsistent with the requirements from the domain model.

Possible consequences of this validation result could include looking for a `checkG` service ("greater") of the `Comparator` component, or changing the business rules slightly, or others. In any case the small but on the domain side non-fictitious validation example has shown that the proposed method gives an early hint at the necessity of a respective, aware decision and provides tangible support for it.

4. Related work, results achieved, and next steps

4.1. Related work

Component software testing theory has become a large area of scientific research [33]. Important existing approaches with relation to the proposed method have been selected (Table 3) to demarcate original contributions of the proposed method. Lines in Table 3 list the examined approaches which are further described below. Columns list three abstraction levels:

component, composition and context. On component level formal program verification of single components with their interfaces is typical research focus. On composition level research from formal and less formal areas deals with architectures of several integrated components. On context level research focus is on the requirements side and less formal, concerned with system architectures in their socio-technical domain and business context.

Approach	**Component (Program)** verification	**Composition (Architecture)** ver. & val.	**Context (Domain)** validation
Built-in test technology	X		
Formal methods	X	(X)	
Scenario-/model-based testing	(X)	(X)	(X)
Specification matching	X		(X)
Tabular notation	X		
Test / composition languages	(X)	X	
Test input data sampling	X	(X)	
Test output oracles	X		

Table 3. Related approaches

The availability of an approach for different abstraction levels is indicated in the cells. The proposed method's research contributions on the domain level or context level are embedding into a clear business model, independent domain based test oracles, and early domain level testing before software is available. The analyzed existing approaches don't seem to cover this.

Built-in test technology. Technologies for self-testing software components, in analogy to built-in tests from integrated circuits, have extensively been researched, e.g. in the Component+ project of the European Union [9]. Built-in tests come within the component, e.g. as additional test services, and are not intended to represent independent customer specific automation requirements but basic technical checks. Tests built into the component by their vendors are complementary to the proposed domain centric axiom.

Formal methods. Especially in formal model checking, plenty of verification approaches ("are we building the software right?") are discussed, among them the interesting domain reduction abstraction [7]. The method proposed here transfers some of the ideas to the domain validation ("are we building the right software?") viewpoint. But fully formal approaches for real components are prevented by computational effort with real systems in practice, decidability problems from computer theory, the absence of complete formal specifications, and the lack of a justifying business case or public interest. Formal verification methods provide valuable insight but in a practical sense don't apply to the proposed method's complex domain level validation.

Scenario based and model based testing. Scenarios can be seen as special entities within the more general notion of a model. In model based testing, test references are generated from a model of the actual system. Many model based test approaches build upon the UML (Unified Modeling Language) today, and derive test references from UML diagrams [6,23]. Test references in existing approaches are built from artifacts within the component software development – models, design scenarios, etc. – and not from independent and unknown customer requirements as proposed in this research. Few if any approaches have yet addressed these model independency issues and test implications as does the proposed method on the domain validation side.

Specification matching. Existing approaches are based on fully formal language specifications, focus strongly on technical aspects, and are restricted to the matching of relatively simple functions [19,36]. Semi-formal matching methods from library science have also been described since long [26,28], and discussions exist to automatically extract classification attributes from natural language descriptions [14]. Further investigations include in particular relaxations of exact matching, and also contextual refinement theory [10]. Only recently did discussions with focus on more complex business domain perspectives start with compatibility considerations of multi-layered specifications [38]. The proposed method goes beyond formal technical aspects and validates specifications vs. higher-order domain level requirements by comparing them with complex domain scenarios.

Tabular notation. This approach aims at representing requirements fully formal by using a comprehensible, mathematically precise tabular notation of predicate logic for partial functions [24]. Domain requirements are successively translated into this tabular form with promising practical results [2]. Tabular notation seems very formal for "good enough" validation testing as intended in the proposed method.

Test and composition languages. Similar to well known semi-formal specification languages such as Z or OCL, special languages for testing and for composition have been proposed. One example is TTCN-3 for test execution [12]. Another example is the Piccola calculus for formal component composition [1]. Test languages make implicit assumptions on their domains and their intended use, and have proven successful for testing software in their respective target

areas. Architectural composition languages are formal and powerful but don't seem suitable for defining and evaluating actual test scenarios. The proposed method suggests a generic, widely applicable domain validation method without actual software but based on reuse specifications.

Test input data sampling. Exhaustive testing on all possible inputs is infeasible in general and inappropriate in particular for large real life enterprise applications. Hence an incomplete but appropriate test has to be determined. Existing approaches achieve this by sampling a domain of the input data according to fault hypotheses i.e. assumptions about which aspects or entities are error prone, allowing the test to reveal as many failures as possible with a minimum effort [3]. In the proposed method, validation tests are generated not from fault hypotheses within the technological software system or its specification or models, but instead independently from the actual customer's ontological domain and its automation requirements which are unknown to, and detached from, the component software technology provider.

Test output oracles. The test oracle question [30,34] relates to outputs produced by a test: if the actual results differ from the expected results, did a proper test run produce wrong results revealing a software error, or were the expected results and/or the testing and/or basic assumptions wrong in the first place? Particular test outputs need careful analysis if the oracle grounds on the same model as the software [27]. Related issues are associated with the attempts of N-version programming in the 1980s. Sophisticated approaches such as e.g. [13] exist today. The proposed method sets priority to tests created independently from a software user, to deliver the oracle and the final judgment about an expected feature of a component.

4.2. Results achieved and next steps

A domain level approach supporting early higher-order black box component software validation on the domain level is described, embedded into a business scenario, and performed in a small example which is non-fictitious on the domain side. The proposed enterprise validation testing method includes abstracting, reducing and including a domain model into validation scenarios. The resulting scenarios are branch-free paths of automation demands deemed validation critical on the domain level. The validation scenarios represent references against which relevant levels from multi-dimensional supplier black box specifications are validated very early in the compositional development process, and with oracles that are independent from this development process.

Next research steps include the identification, evaluation and development of actual end user criteria for reduction and inclusion, and the analysis of multi-layered specification approaches, and related coverage measures. It is expected to lead to the identification of specification defaults or, patterns, and to idioms for the generated validation scenarios. Finally, a proof of concept from a real industry project, possibly building a supporting software tool, is intended.

5. Summary and conclusion

Compositional reuse for industry style software production is an important approach pursued to master the ever increasing demands on software intensive enterprise sized systems. Testing black box software components from large repositories for suitability in an actual end user situation is among the problems that complicate this approach. The associated validation activities are supported by the presented method, offering to the component demand side a domain centric component validation approach which is embedded into a clear business scenario, which sources test oracles from business domain requirements independent from the technological development process, and which produces tangible results early, before the executable software is available, on the basis of reuse specifications. Core elements are customer domain models that become abstracted, reduced and included for validation against multi-layered supplier specifications.

The approach will benefit software component customers through earlier and better validation testing within further industrialized software engineering processes.

References

[1] Achermann, F., Nierstrasz, O.: "A calculus for reasoning about software composition", *Theoretical Computer Science* (331)2-3: 367-396, 2005.

[2] Baber, R. L., Parnas, D. L., Vilkomir, S. A., Harrison, P., O'Connor, T.: "Disciplined Methods of Software Specification: A Case Study". In International Symposium on Information Technology: Coding and Computing (ITCC 2005), pp. 428-437, Las Vegas, USA, 4.-6. Apr., 2005.

[3] Beizer, B.: *Black-Box Testing: Techniques for Functional Testing of Software and Systems*. John Wiley & Sons, New York, 1995.

[4] Biggerstaff, T., Richter, C.: "Reusability Framework, Assessment, and Directions", *IEEE Software* (4)2: 41-49, 1987.

[5] Boehm, B. W.: "The Future of Software Processes". In Unifying the Software Process Spectrum: International Software Process Workshop (SPW 2005): Revised

Selected Papers, pp. 10-24, Beijing, China, 25.-27. May, 2006.
[6] Briand, L., Labiche, Y.: "A UML-Based Approach to System Testing", *Journal of Software and Systems Modeling* (1)1: 10-42, 2002.
[7] Choi, Y., Heimdahl, M.: "Model Checking Software Requirement Specifications using Domain Reduction Abstraction". In 18th IEEE International Conference on Automated Software Engineering (ASE 2003), pp. 314-317, Montreal, Canada, 6.-10. Oct., 2003.
[8] Dietz, J.: *Enterprise Ontology: Theory and Methodology*. Springer, Berlin, 2006.
[9] Edler, H., Hörnstein, J.: "Component+ Final report 1.1." http://www.component-plus.org/pdf/reports/Final report 1.1.pdf, download 12-Oct-2005.
[10] Fidge, C. J.: "Contextual Matching of Software Library Components". In 9th Asia-Pacific Software Engineering Conference (APSEC'02), pp. 297-306, Gold Coast, Queensland, Australia, 4.-6. Dec., 2002.
[11] Gordijn, J., Akkermans, H.: "Designing and Evaluating E-Business Models", *IEEE Intelligent Systems* (16)4: 11-17, 2001.
[12] Grabowski, J., Hogrefe, D., Réthy, G., Schieferdecker, I., Wiles, A., Willcock, C.: "An introduction to the testing and test control notation (TTCN-3)", *Computer Networks* (42)3: 375-403, 2003.
[13] Hummel, O., Atkinson, C.: "Automated Harvesting of Test Oracles for Reliability Testing". In 29th Annual International Computer Software and Applications Conference (COMPSAC 2005), pp. 196-202, Edinburgh, Scotland, UK, 26.-28. Jul., 2005.
[14] Maarek, Y. S., Berry, D. M., Kaiser, G. E.: "An Information Retrieval Approach For Automatically Constructing Software Libraries", *IEEE Transactions on Software Engineering* (17)8: 800-813, 1991.
[15] McIlroy, M. D.: "Mass Produced Software Components". In Software Engineering: Report on a conference sponsored by the NATO Science Committee, pp. 138-155, Garmisch, Germany, 7.-11. Oct., 1968.
[16] Meyer, B.: "Applying "Design by Contract"", *IEEE Computer* (25)10: 40-51, 1992.
[17] Meyer, B.: "The Grand Challenge of Trusted Components". In 25th International Conference on Software Engineering (ICSE 2003), pp. 660-667, Portland, Oregon, USA, 3.-10. May, 2003.
[18] Mili, H., Mili, F., Mili, A.: "Reusing Software: Issues and Research Directions", *IEEE Transactions on Software Engineering* (21)6: 528-562, 1995.
[19] Moormann Zaremski, A., Wing, J. M.: "Specification Matching of Software Components", *ACM Transactions on Software Engineering and Methodology* (6)4: 333-369, 1997.
[20] Myers, G. J.: *The Art of Software Testing*. John Wiley & Sons, New York, 1979.
[21] Object Management Group (Ed.): "UML 2.0. OCL Specification". http://www.omg.org/cgi-bin/apps/doc?formal/06-05-01.pdf, download 24-Oct-2006.
[22] Object Management Group (Ed.): "Unified Modeling Language: Superstructure version 2.0". http://www.omg.org/docs/formal/05-07-04.pdf, download 31-Mar-2006.
[23] Offutt, A. J., Abdurazik, A.: "Generating Tests from UML Specifications". In The Unified Modeling Language: Beyond the Standard: Second International Conference (UML'99), pp. 416-429, Fort Collins, Colorado, USA, 28.-30. Oct., 1999.
[24] Parnas, D. L.: "Predicate Logic for Software Engineering", *IEEE Transactions on Software Engineering* (19)9: 856-862, 1993.
[25] Parnas, D. L.: "Software Aspects of Strategic Defense Systems". In Hoffman, D. M., Weiss, D. M. (eds.): *Software Fundamentals: Collected Papers by David L. Parnas / edited by Daniel M. Hoffmann and David M. Weiss*, pp. 497-518. Addison-Wesley, 2001.
[26] Penix, J., Alexander, P.: "Efficient Specification-Based Component Retrieval", *Automated Software Engineering* (6)2: 139-170, 1999.
[27] Pretschner, A., Philipps, J.: "Methodological Issues in Model-Based Testing". In Broy, M., Jonsson, B., Katoen, J.-P., Leucker, M., Pretschner, A. (eds.): *Model-Based Testing of Reactive Systems: Advanced Lectures*, pp. 281-291. Springer LNCS, Berlin, 2005.
[28] Prieto-Díaz, R., Freeman, P.: "Classifying Software for Reusability", *IEEE Software* (4)1: 6-16, 1987.
[29] Szyperski, C.: *Component Software: Beyond Object-Oriented Programming*. Addison-Wesley, Harlow, 1998.
[30] Turing, A.: "Systems Of Logic Based On Ordinals", *Proceedings of the London Mathematical Society Ser. 2* (45): 161-228, 1939.
[31] Turowski, K. (ed.): *Standardized Specification of Business Components*. GI Working Group 5.10.3 Component Oriented Business Application Systems, Augsburg, Germany, 2002.
[32] Turowski, K.: *Fachkomponenten: Komponentenbasierte beriebliche Anwendungssysteme*. Shaker, Aachen, 2003. In German.
[33] Vincenzi, A. M. R., Maldonado, J. C., Delamaro, M. E., Spoto, E. S., Wong, W. E.: "Component-Based Software: An Overview of Testing". In Cechich, A., Piattini, M., Vallecillo, A. (eds.): *Component-Based Software Quality: Methods and Techniques*, pp. 99-127. Springer LNCS, Berlin, 2003.
[34] Weyuker, E. J.: "On Testing Non-testable Programs", *The Computer Journal* (25)4: 465-470, 1982.
[35] Weyuker, E. J.: "Testing Component-Based Software: A Cautionary Tale", *IEEE Software* (15)5: 54-59, 1998.
[36] Yellin, D. M., Strom, R. E.: "Protocol Specifications and Component Adaptors", *ACM Transactions on Programming Languages and Systems* (19)2: 292-333, 1997.
[37] Yourdon, E.: "When good enough software is best", *IEEE Software* (12)3: 79-81, 1995.
[38] Zaha, J.: "Automated compatibility tests for business related aspects of software components". In On the Move Federated Conferences (OTM 2004) PhD Symposium, pp. 834-841, Larnaca, Cyprus, 25.-29. Oct., 2004.

Model-based Empirical Performance Evaluation based on Relational Traces *

Marko Bošković
marko.boskovic@informatik.uni-oldenburg.de
TrustSoft Graduate School
Carl von Ossietzky University, Oldenburg

Abstract

Empirical performance evaluation is the process of measuring and calculating performance metrics of deployed software systems. It is a part of performance validation during testing of a software system. The topic of this thesis is an approach for the empirical performance evaluation in the context of model-driven engineering. The hypothesis which will be examined is whether concepts of the temporal databases theory can be used as a general way for empirical performance evaluation of model-driven developed software.

1 Introduction

Since the introduction of software engineering, raising complexity of software systems is a persistent problem. High complexity of software makes development harder, and may lead to an increased number of failures. To solve this problem, several attempts like structured, object-oriented and component-based software engineering were introduced. However, the problem still persists. Model-driven engineering (MDE) [10] is an approach which deals with the software complexity by making software models primary artifacts of software development.

A model is a set of statements about the system under study [25]. There are several benefits of this kind of approach but the most important is that a model is much closer to the problem domain then to the underlying implementation. It moves the focus of software engineering from technology specific implementation to the problem domain.

One approach for MDE is *Model Driven Architecture (MDA)* [16]. The MDA suggests a definition of non-proprietary standards which define interoperable technologies for the realization of model-driven engineering [20]. It also suggests the usage of the Meta-Object Facility (MOF) [17] for the specification of modeling languages.

To provide trustworthy software, quality attributes [7] have to be satisfied. Performance is a quality attribute which expresses the degree to which a software system or its component meet its objectives for timeliness [27].

Currently, in model-driven engineering most of the research is dedicated to performance prediction with mathematical analysis methods, or with simulation [1][19]. Nevertheless, predictions have to be validated when the software system is implemented and deployed. A validation should be based on modeling constructs as predictions are made. However, the timing behavior is currently observed in the terms of source code constructs (e.g., method execution time). In MDE the level of abstraction is raised, and models are the primary artifacts of software development. In this case, timing behavior observation of software based on source code constructs is not appropriate. Observations should be done based on modeling constructs (e.g., states, activities, methods or domain specific constructs).

This thesis introduces a general approach for empirical performance evaluation in model-driven developed software. The approach can be applied to any modeling language used for software development. It is based the temporal databases theory [33]. The temporal databases theory relates facts stored in a relational manner with time information. For this reason, the approach presented here suggests the empirical evaluation based on relational traces. A program trace is a dynamic list of events generated by the program as it executes [14]. In relational traces data about identifying an event and timing information about it are stored in sets of tuples, or tables, in terms of databases. The

*This work is supported by the German Research Foundation (DFG), grant GRK 1076/1

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

method will be applied in the context of the MDA approach for MDE.

The contribution of the thesis is a definition of a basic package for defining instrumentation languages. In software engineering, instrumentation is the process of adding software probes to a program [27]. Software probes are additional pieces of code for collecting data about software execution. From the basic package, a language for instrumentation of some particular modeling language can be derived. Derived instrumentation languages enforce data collection in a relational manner. As examples, two instrumentation languages will be derived, for the UML class diagram modeling language and for the UML state machine modeling language. Furthermore, a trace analysis query language will be developed. This analysis language will provide ability to calculate performance metrics.

The paper is structured as follows. The next section gives an overview of model-driven engineering and the Model Driven Architecture intiative. The motivation and foundations of the approach are presented in Section 3. Section 4 outlines work packages in the research plan. Related work is discussed in Section 5, and Section 6 concludes the paper.

2 Model-Driven Engineering

Modeling is very important for engineering disciplines such as electrical or civil engineering. Engineers use models to express some statements about a system that already exist or that shall be made.

A model is a set of statements about some system under study [25]. Generally, two kinds of models exist: models of existing systems or models of systems which are to be implemented.

Models as a description of existing systems are used to deduct new statements about those systems, or predict behavior of the system in some particular situation. For example, this kind of models are used in physics or chemistry.

As a specification for systems that shall be developed, models are used in engineering disciplines, such as electrical engineering. In this case, models are used to predict characteristics of the system before it is finally produced.

The general idea of MDE is to introduce the model as a first class entity, as it is in the case of other engineering disciplines. With models, software development gets more closer the problem domain, and not to the implementation. They enable the decomposition of problems in a way which is suited to the nature of the system, or part of it, which we are specifying. Generaly, there are two kinds of systems, reactive and transformational [31]:

- Reactive systems—systems which are highly interactive with the environment. This kind of systems are able to react to the environment change of states, and enforce certain desirable behavior. Examples of this kind of systems are control, real-time and embedded systems. Information systems as well are reactive systems, because they interact with the organization in their environment. The behavior of this kind of systems is specified with a stimulus/response behavior, such as UML state machines [18].

- Transformational systems—systems which exist to transform an input into an output. Routines in mathematical libraries, compilers, assemblers, and queries over databases are examples of transformational systems. These systems are used only for computation, and when the computation is over, the system is terminated. They can interact with the environment, but only for the purpose of calculation of the output, and not for managing a state of it. Specification of this kind of systems is done with techniques for functional decomposition. With functional decomposition, the system functionality is mapped to the internal components of a system and their collaboration, such as UML class and sequence diagrams [18].

Systems can consist of both, transformational and reactive parts. For example, in the Model-View-Controller [6] three-tier architecture, the Controller is a reactive part, because it defines the way the system reacts to user input. Nevertheless, the generation of Views, from Model is the transformational part, because it generates output from the user input.

Beside moving software development to the problem domain, model driven engineering impedes the ability to exploit formal mathematical methods. Furthermore, with abstraction, understandability can be improved, and state explosion problems in applied mathematical methods can be avoided. Finally, because of the nature of software artifacts, a complete implementation can be generated without discontinuities in the expertise, materials, tools or methods [26]. This implies that the primary and the final artifact of software development in the MDE approach is a model.

The Model Driven Architecture (MDA) [16] is one approach for MDE initiated by the Object Management Group (OMG), a consortium of software vendors and users from industry, academia, and government. The MDA initiative consists of three complementary ideas [2]:

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

1. *Direct representation*—Shift the focus of software development away from the technology toward the ideas and concepts of the problem domain.

2. *Automation*—Mechanization of relating semantic concepts of a problem domain and an implementation domain.

3. *Open standards*—Usage of non-propertiery standards that will specify interoperable technologies. These standards will close the semantic gap between domain problems and standard implementation technologies.

The topic of this thesis is to enable an empirical evaluation of software performance when the primary artifact of software development is a model. As the approach for model driven development, the MDA will be used. That imposes the usage of open standards defined by the OMG, MOF [17] and UML [18]. In the next section, the motivation and foundations are presented in more detail.

3 Foundations and Motivation

3.1 Software Performance, Evaluation and Motivation

Performance is in the thesis considered as the degree to which a software system or component meets its objectives for timeliness [27]. It can be evaluated by simulation, analytical modeling or empiricaly [14]:

- Simulation is a program which enables an imitation of a program execution. In the simulation, only important parts of an execution are imitated. It is less expensive and more flexible then building a real system and then empirically evaluating. However, simulation is not as accurate as measuring real systems.

- Analytical modeling is a technique where a system is mathematically described. Results of an analytical model tend to be much less accurate then real system measurements. However, it is the fastes way to get some initial insight into the behavior of the system, or part of it.

- Empirical evaluation is performed by measurements and metrics calculation. This provides the most accurate results, as no abstractions are made.

Currently, there exist approaches for analytical evaluation of software performance from annotated models and for simulation [1]. However, there is still not an approach for empirical evaluation of software performance. The topic of this thesis is to introduce an approach for performance evaluation based on modeling constructs.

A necessary part of empirical performance evaluation is the software execution data collection which is achieved by instrumentation. The next subsection gives an overview of instrumentation.

3.2 Instrumentation

In software engineering, instrumentation is the process of adding software probes to the program [27]. Software probes are additional pieces of code for collecting data about the software execution. They can be developed for different techniques of the software execution data collection. Generally, there are two techniques for collecting data about a program execution, sampling and event tracing:

- Sampling is a technique where parts of a program are sampled during its execution in some time interval. It is a general statistical technique in which a representative sample of data about the program during execution is taken. In this approach, performance overhead due to measurement is constant. However, a general shortcoming is that the data collected, when the same experiment is performed twice, will hardly be the same. Moreover, infrequent events can easily be missed. An example of this kind of approach is sampling the program stack to follow the execution of a program.

- Event tracing is a process of generating traces of events in the software. A program trace is a dynamic list of events generated by the program as it executes [14]. It contains events for characterization the overall program behavior oredered by time. With this technique the processing overhead can be significant in case of very frequent events. Another problem is the size of an event trace. Generally, a trace can be very large if we trace each instruction execution or very frequent events. Large traces occupy resources like memory and, therefore, impact the overall performance of the program.

In the thesis event tracing, as a technique for performance evaluation, will be used. Furthermore, traces will be collected in a relational manner using the concepts of temporal database theory.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

3.3 Temporal Databases

Temporal databases are databases which support some notion of time [33]. Conventional (non temporal) databases in which are stored only facts can provide only a possibility of representing a state of a system. For each fact stored in a temporal database a piece of time time information.

Facts in databases are represented as rows in tables, or tuples as defined in relational algebra. These facts can be related to the valid time dimension and to the transaction time dimension [33]. The valid time dimension is related to the time when the fact was true in reality. The transaction time dimension is related to the presence of the fact in the database.

Temporal databases which store only facts about the past are called historical databases [33]. Historical databases define two kinds of relations, event and interval relations [24]. Interval relations are used for storing facts which were true for some time interval. Event relations are used for storing facts which were true at some particular point in time. Those facts were valid only at that time point.

One hypothesis the approach is that all parts of an execution of a program defined by modeling elements of a particular language can be characterized as one of the two previously mentioned kinds of facts.

3.4 Problem, Hypothesis and Conceptual Structure of the Solution

The problem and challenge of this thesis is to raise the level on which performance evaluation is done. Currently, evaluations are performed at the implementation technology level, and mostly based on code constructs. In model-driven developed software, observations of behavior should be done in terms of modeling constructs. Instrumentation, for observing software in the terms of modeling constructs and on the code level would require knowledge of the transformation. For example, if a system is modeled with UML state machines, and assume that implementation is in Java. In that case, instrumentation for observation of timing behavior in the terms of states would require knowledge of the implementation strategy of statecharts. Despite the fact that this is not according to the idea of model-driven engineering, it can be error-prone. Therefore, instrumentation should be done at the model level.

According to the nature of a part of a system, transformational or reactive, appropriate formalisms are used for modeling, as explained in the previous section. Therefore, a system specification uses several modeling languages. The challenge is to introduce the evaluation an approach suited for different modeling formalisms.

The hypothesis of this thesis is that concepts of the temporal database theory can be applied for a performance evaluation approach in the context of model-driven engineering, for all modeling language. With the usage of temporal database theory, we can use already existing implementations of databases for various analyses. Furthermore, we assume that this easily leads to a declarative calculation of performance metrics. As examples for stimulus/response and functional decomposition languages, UML state machines and class diagrams will be used.

The conceptual structure of the solution is presented in Figure 1.

The basic instrumentation package will give the basic structure of traces and the definition of temporal data which can be collected, based on the temporal database theory. The most important concepts in this basic package are interval trace and event trace.

In the case when some element of a modeling language models a part of an execution of the program which lasts for some time interval, it will be instrumented by specialization of the interval trace. Specializations of event traces are used for an instrumentation of parts of executions which only occur at some point in time and do not have a duration. The basic trace types are specialized for each modeling language separately, depending on the type of modeling constructs and the time characterisation of their semantics. For example, in the case of UML class diagrams, there would be only one trace kind, for instrumentation of method calls, and it would be derived from the interval trace. However, in the case of UML state machines there will be event traces for instrumentation of state machines. An event trace will be specialized for instrumentation of outgoing and incoming signals, and for instrumentation of the states, the interval trace will be specialized. The initial structure of the basic instrumentation package is out of scope of this paper.

The trace analysis query language will be also developed. Foundations for this language will be temporal query languages and performance metrics, and it will be general for any instrumentation language.

4 Research Plan

The research plan is divided into the following five work packages:

1. **The basic instrumentation metamodel package development**
   The outcome of this work package is a MOF-based basic package for instrumentation. During this

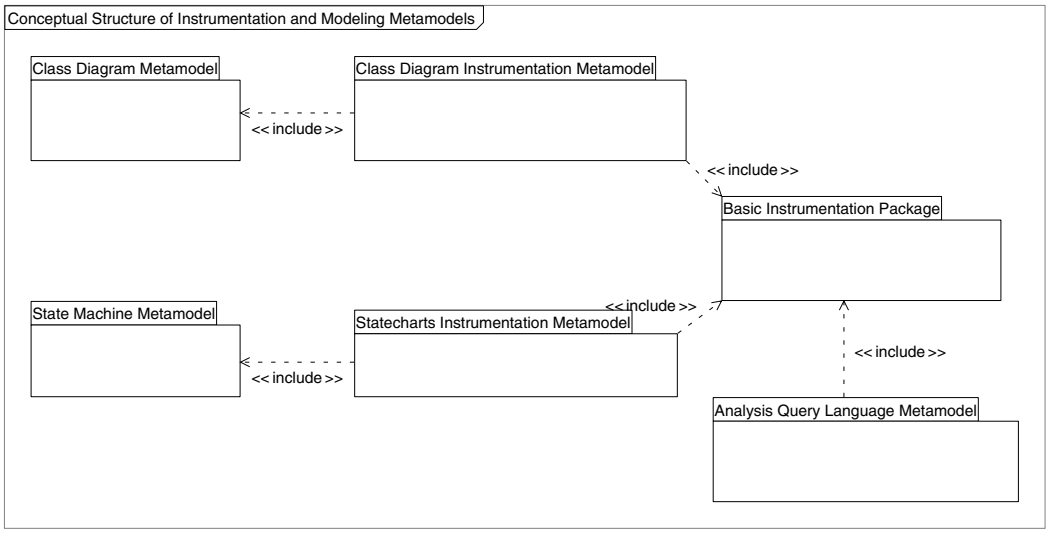


Figure 1. Conceptual structure of the solution

work package the metrics for performance evaluation will be defined, and the basic instrumentation package will be developed according to them. The methodology which will be used is a survey. The survey analyses existing literature. This work package is currently in progress.

2. **An instrumentation language for a UML class diagrams modeling language development**
   The outcome of this work package is an instrumentation language and a transformation for a UML class diagram based modeling language. In this work package, an MOF based abstract syntax of the language, a concrete syntax in the form of a UML profile, and transformations from a model to Java code will be developed. This work package is in an early phase.

3. **An instrumentation language for a UML state machine modeling language development**
   The outcome of this work package is an instrumentation language and a transformation for a state machine based modeling language. This work package, as the previous, will provide an MOF based abstract syntax, a concrete syntax in the form of a UML profile, and transformations from model to Java code. This work package is in an early phase.

4. **The traces analysis language development**
   The outcome of this work package is a query language for trace analysis. As in the previous cases, the abstract syntax as an MOF defined metamodel. A concrete syntax will text based, and transformations from model to code for analysing traces will be developed.

5. **Feasibility evaluation**
   For a feasibility evaluation, the ArcStyler (`http://www.interactive-objects.com/products/arcstyler`) will be used. In the evaluation, instrumentation languages for Accessors, web tier components for three tier architectures will be developed, and for Components, business object models. Furthermore, transformations from model to code will be developed. Finally, impact of measurements on performance of original code will be done. The feasibility evaluation will be done using the JEE "Pet Store" (`http://java.sun.com/developer/releases/petstore/`) case study and the JBoss (`http://www.jboss.org`) application server.

5 Related Work

At the moment there are several approaches for analytical evaluation of software performance from annotated models [19][23] and for simulation [21]. A detailed survey related to performance prediction can be found in Balsamo et al. [1]. However, there are several approaches for measurement and instrumentation in the context of code-centric development.

Snodgrass [29] introduces a relational approach for monitoring systems. His work showed that relational

data structures can be an appropriate formalism for monitoring dynamic behavior of a system. According to parts of a system, relations are specified. Furthermore, TQuel [28] is introduced, a language for querying the system state and filtering relational based event traces. In Snodgrass's approach, the programmer manually defines the instrumentation according to concepts of an existing system. Our approach provides a schema for defining instrumentation languages according to modeling formalisms used for the specification of programs. Furthermore, TQuel provides queries which enable only observation of temporal relations between stored facts. Queries in the query language for performance analysis will enable a calculation of a performance metric through an extension with aggregate functions over temporal data.

Liao et al. [13] introduce a high-level language for program instrumentation and monitoring. In their language, a programmer specifies monitoring and measuring questions. According to the specification, the static analysis of code is done and instrumentation code is added. Conceptualy the approach uses a relational approach for the representation of data collected during a program execution. Nevertheless, the language which is developed is only suited for functional decomposition of systems. The approach which is the topic of this thesis shows that temporal relations can be used as a general approach independent of formalism used for decomposition of a software system.

One more language for program instrumentation is the Metric Description Language (MDL), introduced by Hollingsworth et al. [8] as a part of the Paradyn Parallel Performance Tool. The MDL has the ability to define instrumentation as a separate concern, independent of the functionality of a program, define points at which actions for measurement should take place and intertween it with the program during runtime. Points where measurements can take place are procedure entry, procedure exit and individual call statements. Therefore, as in the previous case the langauge is suited for functional decomposition only.

The Application Response Measurement (ARM) [30] is a technical standard for response times and for a provision of status of transactions in business applications. It provides a technology independent specification for measurements and implementation can be done in different languages. This standard specifies only a framework for performing measurements, but does not define how to analyse collected data. The methodology in the thesis defines both, how to perform measurements and how to analyze collected data.

Aspect Oriented Programming (AOP) is a programming approach which can be used for transparent software instrumentation. Transparent means that the source code of the software functionality is not mixed with probes. Examples for the application of AOP for an instrumentation are introduced by Marenholtz et al. [15] and Debusmann et al. [3]. Debusmann et al. [3] combine AOP with the ARM standard. Marenholz et al. [15] use AspectC++ for an instrumentation of an operating system for debugging, profiling/measurement, and runtime surveillance/monitoring.

For a transparent instrumentation of CORBA-based component systems, interceptors can be used. Interceptors are similar to AOP and can intercept method invocations to transparently instrument the program. Examples of the usage of portable interceptors in instrumentation are presented by several authors. For example, Li [12] introduces an approach where software probes are predefined and placed in stubs and skeletons during an IDL compilation. The measurement probes can be turned on and off at runtime. Probes are inserted in a program with a modified compiler, which is another technique for transparent instrumentation. Debussmann et al. [4] combine an ARM library and portable interceptors for software instrumentation.

Instrumentation can also be done by adding a transparent software layer to the application. Diaconescu et al. [5] introduce an approach where, at deployment time, a transparent proxy layer for the collection of execution data is automatically generated.

ProbeMeister [22] is a tool for runtime Java bytecode instrumentation. It has an extensible set of software probes that can be inserted or removed at any point while an application is running. In the approach introduced by this tool, the Java Debugging Interface is used for runtime instrumentation. ProbeMeister is a separate application and can be used for instrumentation of multiple JVM's.

For collecting software execution data, the underlying platform can be instrumented. This kind of approach is presented by Yeung et al. [32]. The authors developed a virtual JVM called Verneer which is a Java program running on the top of the original JVM. The virtual JVM is a software layer which intercepts class loading, and fragments methods. Each method can be fragmented at the point of a method call, a method entry, basic blocks, and return. After the fragmentation, probes are added to these fragments. Another way to instrument the program is by using a pre-instrumented platform. A pre-instrumented platform is a platform with integrated instrumentation, like Jinsight [9].

Aspect Oriented Programming, portable interceptors, addition of a transparent software layer, bytecode

instrumentation, and underlying platform instrumentation are approaches that enable a collection of data about software execution in a way that the basic functionality code is not influenced. The approach presented in the thesis enables a collection of data about the software execution on a model level, and as a separate concern. Furthermore, it is a performance evaluation approach and not only an instrumentation approach.

Klar et al. [11] developed a set of tools for model-driven instrumentation. They define different sets of models for a program, a functional program model, a functional implementation model, a performance model and a monitoring model. In a functional program model, they explicitly model the functional interdependence of activities without implementation details. A functional implementation model is a detailed model which is concerned with the implementation of the functional program model. A performance model supports validation and is a prerequisite for predicting the performance of not yet implemented programs. In this model, realistic time attributes should be assigned according to measurements performed. Finally, a monitoring model is a subset of the functional implementation model, with which measurement points are defined. Measurements are performed according to the chosen level of abstraction. The source code is instrumented according to this monitoring model. For an analysis tool, a trace description is generated. With a generated trace description, an evaluation tool can decode an event trace. In the approach presented in this paper, the primary artifact of software development is a model. Therefore, instrumentation is done on the model level. The functional implementation model is actually a product of software development. Furthermore, instrumentation defines what to measure and where to measure, and from these two models, source code is generated.

6 Conclusion

Empirical performance evaluation enables validation of timeliness of a software system. However, so far there exist no approach for empirical performance evaluation in the software development process where a model is the primary software artifact.

The paper outilnes, a research with the purpose to give a general approach for empirical performance evaluation of model-driven developed software. This kind of software development is called model-driven engineering. Instrumentation and empirical performance evaluation at the moment are done based on implementation language constructs. Instrumentation on the implementation technology level for collecting data about a program execution in the terms of modeling constructs can be error prone and can require significant effort. Therefore, the instrumentation is done at the model level. The models for software functionality definition and instrumentation definition are separated to reduce the complexity of models.

The contributions of the thesis are a basic package for the definition of instrumentation languages of UML based models, a methodology for deriving instrumentation languages, and a query language for performance metrics calculation. The languages for instrumentation enable data collection in terms of the modeling language constructs, and structured in relational manner.

Evaluation will be based on the Eclipse modeling tool. UML state machines and class diagram modeling languages, will be used as examples of different decomposition techniques. According to the literature survey on temporal databases and performance, and during development of the instrumentation languages for these two languages, the structure of the basic instrumentation package will be developed. Development of the basic instrumentation package will be followed by the development of a query language for trace analysis.

The feasibility of the approach will be done by integrating software performance evaluation based on relational traces in the ArcStyler commercial tool for model-driven development. Furthermore, experiments on the JEE "PetStore" case study will be performed to show the impact of instrumentation code and execution on the instrumented application.

References

[1] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simeoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering*, 30(5):295–310, 2004.

[2] G. Booch, A. Brown, S. Iyengar, J. Rumbaugh, and B. Selic. An MDA Manifesto. *In The MDA Journal: MDA Straight from the Masters*, November 2004.

[3] M. Debusmann and K. Geihs. Efficient and Transparent Instrumentation of Application Components using an Aspect-oriented Approach. In *14th IFIP/IEEE Workshop on Distributed Systems: Operations and Management (DSOM 2003)*, volume 2867 of *LNCS*, pages 209–220, Germany, October 2003. Springer.

[4] M. Debusmann, M. Schmid, and R. Kroeger. Measuring End-to-End Performance of CORBA Applications using a Generic Instrumentation Approach. In *ISCC '02: Proc. of the 7th Int. Symp. on Computers and Communications*, pages 181–186, Washington, DC, USA, 2002. IEEE Computer Society.

[5] A. Diaconescu, A. Mos, and J. Murphey. Automatic Performance Management in Component Based Systems. In *Proc. of the 1st Int. Conf. on Auton. Comp. ICAC'04*, pages 214–221. IEEE Comp. Society, 2004.

[6] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, Boston, MA, USA, 1995.

[7] W. Hasselbring and R. Reussner. Toward trustworthy software systems. *Computer*, 39(4):91–92, 2006.

[8] J. K. Hollingsworth, O. Niam, B. P. Miller, Z. Xu, M. J. R. Goncalves, and L. Zheng. MDL: A Language And a Compiler For Dynamic Program Instrumentation. In *PACT '97: Proc. of the 1997 Int. Conf. on Parallel Archit. and Compil. Techniq.*, pages 201–213, Washington, DC, USA, 1997. IEEE Comp. Society.

[9] IBM. Jinsight. web: http://www.alphaworks.ibm.com/tech/jinsight, 1998. Accessed November 2006.

[10] S. Kent. Model Driven Engineering. In *IFM '02: Proc. of the 3rd Int. Conf. on Integrated Formal Methods*, pages 286–298, London, UK, 2002. Springer-Verlag.

[11] R. Klar, A. Quick, and F. Soetz. Tools for a Model–driven Instrumentation for Monitoring. In *Proc. of the 5th Int. Conf. on Modelling Techniques and Tools for Computer Performance Evaluation*, pages 165–180. Elsevier Science Publisher B.V., February 1991.

[12] J. Li. Monitoring of Component-Based Systems. Technical Report HPL-2002-25, Imaging Systems Laboratory, HP Laboratories Palo Alto, 2004.

[13] Y. Liao and D. Cohen. A Specificational Approach to High Level Program Monitoring and Measuring. *IEEE Trans. on Software Engineering*, 18(11):969–978, 1992.

[14] D. J. Lilja. *Measuring Computer Performance: A Practitioner's Guide*. Cambridge University Press, Cambridge, UK, 2000.

[15] D. Mahrenholz, O. Spinczyk, and W. Schroeder-Preikschat. Program Instrumentation for Debugging and Monitoring with AspectC++. In *Proc. of the 5th Int. Symp. on Object-Oriented Real-Time Distributed Computing ISORC '02*, pages 249–256, Washington, DC, USA, 2002. IEEE Computer Society.

[16] J. Miller and J. Mukerji. MDA Guide (Version 1.0) OMG document formal/03-06-01. web: http://www.omg.org/docs/omg/03-06-01.pdf, May 2003. Accessed November 2006.

[17] Object Management Group. MOF 2.0, OMG document ptc/04-10-15. web: http://www.omg.org/cgi-bin/apps/doc?ptc/04-10-14.pdf, October 2004. Accessed November 2006.

[18] Object Management Group. UML 2.0 Specification: Superstructure, OMG document ptc/05-07-04. web: http://www.omg.org/cgi-bin/doc?formal/05-07-04, November 2004. Accessed November 2006.

[19] Object Management Group. UML Profile for Schedulability, Performance, and Time Specification, OMG document formal/05-01-02. web: http://www.omg.org/cgi-bin/apps/doc?formal/05-01-02.pdf, January 2005. Accessed November 2006.

[20] Object Management Group. MDA Guide. web: http://www.omg.org/cgi-bin/doc?ormsc/06-06-02.pdf, 2006. Revision Draft, Accessed November 2006.

[21] D. Park and S. Kang. Design phase analysis of software performance using aspect-oriented programming. In O. Aldawud, G. Booch, J. Gray, J. Kienzle, D. Stein, M. Kandé, F. Akkawi, and T. Elrad, editors, *The 5th Aspect-Oriented Modeling Workshop In Conjunction with UML 2004*, October 2004.

[22] P. Pazandak and D. Wells. ProbeMeister: Distributed Runtime Software Instrumentation. web: http://www.cs.uni-bonn.de/ gk/use/2002/sub/pazandak-ProbeMeister.pdf, 2002. Accessed November 2006.

[23] D. B. Petriu and M. Woodside. A metamodel for generating performance models from UML designs. In *Proceedings of the 7th Int. Conf.: The Unified Modelling Language: Modelling Languages and Applications*, volume 3273 of *LCNS*, pages 41–53. Springer-Verlag, 2004.

[24] N. Sarda. Extensions to SQL for Historical Databases. *IEEE Transactions on Knowledge and Data Engineering*, 02(2):220–230, 1990.

[25] E. Seidewitz. What Models Mean. *IEEE Software*, 20(5):26–32, 2003.

[26] B. Selic. A Short Course on MDA Specifications. INFWEST Seminar on Model Driven Software Engineering. Pirkkala, Tampere, Finland, August 2006.

[27] C. U. Smith and L. G. Williams. *Performance Solutions: A Practical Guide to Creating Responsive, Scalable Software*. Addison-Wesley, Massachusetts, Boston, USA, 2001.

[28] R. Snodgrass. The temporal query language tquel. *ACM Trans. Database Syst.*, 12(2):247–298, 1987.

[29] R. Snodgrass. A Relational Approach to Monitoring Complex Systems. *ACM Transactions on Computer Systems*, 6(2):157–196, 1988.

[30] The Open Group. Application Responce Measurement (ARM). web: http://www.opengroup.org/tech/management/arm, 1998. Technical Standard, Version 2, Issue 4.0, Accessed Novemeber 2006.

[31] R. J. Wieringa. *Design Methods for Reactive Systems: Yourdon, Statemate, and the UML*. Morgan Kaufmann Publishers, San Fransisco, CA, USA, 2003.

[32] K. Yeung, P. H. J. Kelly, and S. Bennett. Dynamic Instrumentation for Java Using a Virtual JVM. In *Performance Analysis and Grid Computing*, pages 175–187, Norwell, MA, USA, 2004. Kluwer Academic Publishers.

[33] C. Zaniolo, S. Ceri, C. Faloutsos, R. Snodgrass, V. S. Subrahmanian, and R. Zicari. *Advanced Database Systems*. Morgan Kaufmann Pubishers, USA, 1997.

The strategic impact of service oriented architectures

Philipp Liegl
Research Studios Austria
Thurngasse 8/20, 1090 Vienna, Austria
philipp.liegl@researchstudio.at

Abstract

It has not been since the advent of the client/server architecture break through that an architectural concept has changed the face of enterprise systems so significantly as it has been done by service oriented architectures (SOA). The service oriented approach provides plenty of vantages for companies in regard to flexible system integration and adoption of new business cases. However, the adoption of SOA in an actual enterprise system brings along a couple of problems as well. Especially the integration into the existing infrastructure, applications and the innovation, sourcing and investment policies is challenging. A solution can be provided by establishing a SOA roadmap unveiling possible traps and pointing out the foibles and flaws still existing in the SOA approach. In this paper the SOA approach will be reviewed critically and the different sections affected within an enterprise will be examined. Possible problems during the transition and use of SOA will be identified. Where already possible, solutions will be provided. This paper is based on current research conducted during my PhD studies.

1. Introduction

Architectures in information systems have seen many changes over the years beginning from monolithic mainframe applications to the latest development - service oriented architectures (SOA). In a service oriented context program functionality is exposed via an interface which is accessible over a network. These interfaces are usually referred to as "services". A service in the context of computer science is an encapsulation of business logic accessible via XML messages over a network and must not be confused with the idea of a service in the context of business administration. In the context of SOA a service is coarse-grained and loosely coupled and it can be reused in several contexts. The lifecycle-management of a service, from its creation until it is abandoned, is part of a service oriented architecture. A specific infrastructure allowing the different applications to exchange data via services is also defined. The most known standards in use for a SOA today are WSDL (Web Service Description Language) [16], SOAP (Simple Object Access Protocol) [17], BPEL (Business Process Execution Language) [1], WS-CDL (Web Service Choreography Description Language) [18] and UDDI (Universal Description, Discovery and Integration) [10] which became de facto standards for a service oriented architecture. Nevertheless, a service oriented architecture must not necessarily be build with these standards. However, the compliance to industry standards, the use of available tools and a maximum of interoperability benefits strongly suggest their use.

The use of XML as the basic format for the exchange of messages and information facilitates the communication between business partners. Typical technical boundaries which prevented B2B eCommerce in the past such as different systems or communication heterogeneity are circumvented by the use of XML. An enterprise planning to adapt

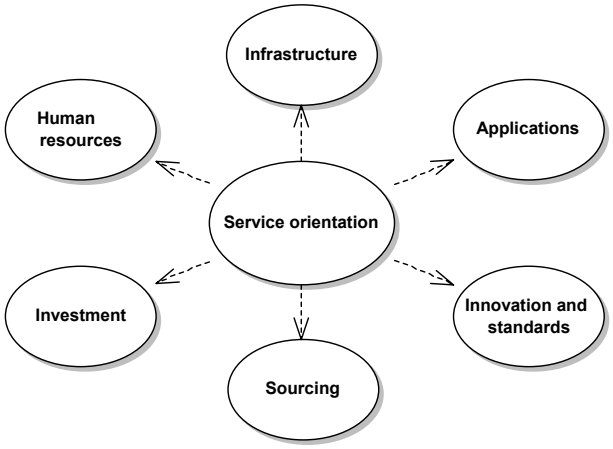


Figure 1. The influence of SOA

service orientation for its IT department has to consider several crucial facts. Looking at the architecture simply from a technical perspective is not enough. The business perspec-

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

tive and the application perspective must also be taken into account. What is needed, is an overall strategy embracing all three perspectives and guaranteeing a successful transition and seamless integration. In order to avoid possible traps which might occur during the transition to service orientation a SOA roadmap should be established. The several sections of an enterprise affected by the installation of a service orientation are covered by such a roadmap. Although no absolute guarantee for success a roadmap greatly facilitates the transition from a classic client/server architecture to a service oriented approach.

2. A roadmap to a service oriented architecture

A transition to SOA will have an influence on several divisions and departments within an enterprise. Figure 1 gives an overview about the different areas being affected by the introduction of service orientation. Infrastructure and applications in use will experience the major changes. However, innovation and standards, sourcing, investment and human resources will also be affected. A service oriented architecture within an enterprise will not remain a self-contained IT-pattern applied to hard- and software, but become a paradigm embracing all parts of the enterprise. The most important changes in every section will be examined in detail.

2.1. Infrastructure

Most of the IT infrastructures in use today are still based on the client/server pattern. The central point of all communication is a server or a mainframe system. These often legacy systems have been around and in use for several years. They are well-tried and the IT staff responsible for them is experienced - failure scenarios are well-rehearsed and occur rarely. The only major flaw these systems have is the inflexibility in regard to extensions and adaptation to new business scenarios. In a constantly faster changing business world such IT systems are not business enhancers but bottle-necks. A solution for more flexibility and faster conformance to new business scenarios is brought by SOA.

Service orientation in a broader sense however, is not really new. With components like CORBA (Common Object Request Broker Architecture) [11] or DCOM (Distributed Component Object Model) [3] service-like approaches were already realized years ago. Nevertheless, these systems were hard-wired and relied on certain standards and operating systems. An adaptation to a new business scenario i.e. through the acquisition of a new company could only be realized with major coding efforts. A real process oriented service composition was not possible.

Within a service oriented approach relevant core program functionality is extracted and offered through an interface. The interface can be accessed via XML messages - so called service providers are created. On the other hand the enterprise's systems also consume other services i.e. from subsidiaries or subcontractors. A service oriented architecture within an enterprise therefore consists of service consumers and service providers. Information about existing services is stored in a so called service registry where the information is made accessible to relevant stakeholders.

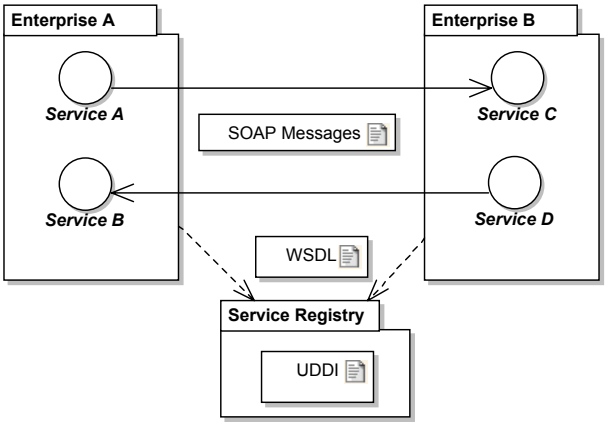


Figure 2. A sample SOA environment

Figure 2 shows a sample SOA environment. Enterprise A has two services - a service consumer (*Service A*) and a service provider (*Service B*). *Service C* of Enterprise B acts as a service provider and *Service D* as a service consumer. For the information exchanged between the different services SOAP messages are used. In order to retrieve information about existing services both enterprises can access a service registry which is based on UDDI. A service registry provides particular functionality, allowing the search and retrieval of relevant service information. Information about existing services is returned in the form of WSDL files. With the use of pertinent software user accessible interfaces can be build on the basis of the WSDL files. The advantage of a service oriented architecture is the flexibility to exchange services. Enterprise A could abandon *Service B* and set up *Service E* instead. In the business case of figure 2 such an exchange must first be communicated to Enterprise B. Any other enterprise willing to consume the new *Service E* however, would know about it because the WSDL information can be retrieved from the service registry.

In reality the use case scenarios of a service orientation are not as simple as described in figure 2. It is necessary to align different service calls, assemble different services to a new service etc. Figure 3 shows a more complex scenario. Enterprise A's *Service A* invokes *Service B* of Enterprise B. *Service B* is a composed service which itself calls two more services namely *Service C* and *Service D*. The orchestration of the calls is achieved via BPEL. With the use of BPEL the

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

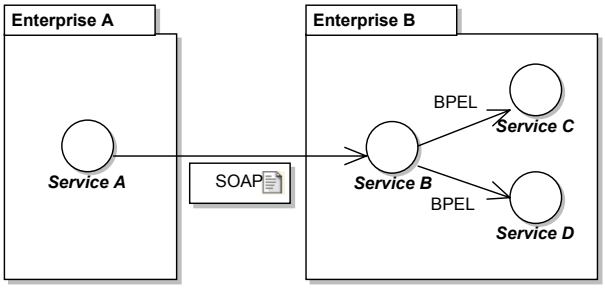


Figure 3. Service orchestration with BPEL

order in which specific services are called can be controlled. In the terminology of web services controlling of service call orders is referred to as *orchestration*. If the sequence of web service calls between two parties is controlled we refer to it as *choreography*. Orchestration must be viewed from a partner specific perspective whereas choreography must be viewed from an overall perspective between two participating parties.

It is not important for Enterprise A what specific software or application is hidden behind *Service B* as long as it serves the needs of Enterprise A. Enterprise B on the other hand is flexible in assembling the logic and functionality of *Service B* by simply changing the BPEL file, orchestrating the calls. The logic hidden behind *Service B* can easily be changed by Enterprise B to fulfill changing process needs required by Enterprise A. *Service B* could also be regarded as a certain interface covering the logic behind it. So far such a concept is not new and well established also in conventional client/server architectures. The major advantage of the SOA approach however, is the possibility to easily assemble a new logic behind *Service B* by simply changing the BPEL definition. Furthermore a new service could be set up by Enterprise B which might be called *Service E* and serves the specific needs of a fictional Enterprise C. *Service E* could reuse the existing services *Service C* and *Service D*. At this point the importance of a service oriented architecture becomes evident - it provides a maximum of flexibility and reuse.

A classic enterprise system based on a client/server architecture does not provide such a flexibility needed for fast integration of new business scenarios. Moreover it now also becomes evident that a pertinent infrastructure is a mission critical factor for implementing a service orientation.

The realization of such an approach requires significant changes in the overall system architecture and infrastructure. In order to guarantee a seamless transition, a SOA roadmap is needed. A SOA roadmap approach consists of four major phases which will be broken down into specific activities: *education*, *assessment and benchmarking*, *planning* and *roll out*. In every phase specific tasks are performed and artifacts are generated which are then used in the next phase. A similar proposal has already been made by SUN Microsystems in [13].

Education: With the increasing popularity of service orientation the perception of its concepts in many cases has became blurry. Two different people talking about SOA often mean two different things. However, a common understanding of the concepts and techniques is a mission critical factor. The business executives and more important the responsible persons within the IT department must have a clear understanding of SOA principles and concepts. Only if a common understanding of the relevant technologies and tools is available, an appropriate assessment and planning of a SOA in the next phases is possible.

Assessment and benchmarking: In this phase the IT representatives must assess the current state of the IT infrastructure within the enterprise. The major question to be answered will be "How ready is the company for SOA?". If possible, service oriented systems in use at subsidiaries or subcontractors should be analyzed in order to get a benchmark for the own future architecture. In particular the question whether to develop the SOA architecture in-house or to buy an external application and customize it to the specific needs must be answered.

Planning: Within this phase a detailed transition plan with the relevant software engineering and project management tasks must be elaborated. A SOA project group should be established, consisting of relevant stakeholders not only from the IT department but from all departments of the company.

Roll-out: In this phase the old architecture is gradually switched off and the service oriented approach is introduced. The transition from the old architecture to the new design is a dynamic process and therefore a vigilant supervision by the SOA project group is necessary.

A major goal of the current research will be the elaboration of a SOA roadmap. In particular the activities of the four phases will be analyzed in detail, while also pointing out possible shortcomings and foibles of the service oriented approach per se.

2.2. Applications

With service oriented architectures a new type of application has evolved as well - so called composite applications. Composite applications are similar to component-based software (CBS) focusing on building large software systems by integrating previously build software components. The nature of a composite application is the fact, that it is build by combining existing services provided by other applications. An example for a composite application is shown in figure 4. *Service B* can be regarded as a composite application, consisting of two services namely *Service C*

and *Service D*. When referring to the terminology used in the last paragraph a composite application is an assembly of service consumers. The newly created functionality of a composite application can then be exposed to other applications consuming its functionality. Hence composite applications can also be seen as service providers. The organization of the service composition is done within a so called composition platform. Again the major advantage is the possibility to exchange the services which are behind *Service B* by simply changing the service orchestration. When viewed in

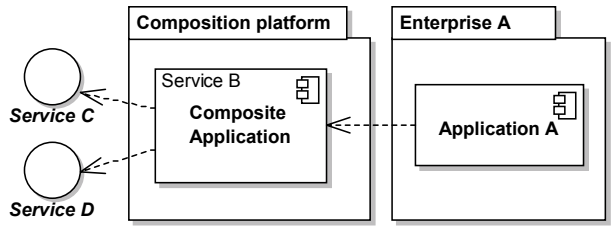


Figure 4. A sample composite application

a coarse-grained fashion, a composite application has three different layers: a user interface layer, a choreography layer and a service layer. Within the user interface layer the different forms and functions are presented to the user. The choreography layer defines the correct order in which the different services the composite application consists of, are called. Most likely BPEL will be the process execution language of choice to perform the orchestration. Services used by the composite application are represented by the service layer. The engineering of such applications will be a major challenge when introducing SOA in an enterprise. One very promising approach is the use of appropriate models to generate the software artifacts.

Application development: One major advantage promised by SOA is the rapid development of new service-based applications. Classic software development is often too complex and development of extensions or the adaptation to new business scenarios takes too long. Knowledge about the software engineering process however, is well accepted and known in the professional world though often only to programmers. In order to facilitate the creation of composite applications a model driven development (MDD) approach is helpful. On the first layer the user must be assisted by the MDD tool in creating a UI for the application. After the UI layer is finished the modeler must assign functionality to the various UI elements. This is done by connecting the UI layer to the processes represented by the second layer. On the second layer the different business processes are orchestrated in a specific order. The user retrieves the available services from a registry. Services can then be dragged and dropped onto a canvas of a modeling tool and be connected according to the specific needs. It is important to notice that the data returned by one service must not necessarily match the data required by another service. A solution is provided in the form of a so called "data mapper", allowing to transform the output of one service into the correct input of another service. The transformation will mostly be done from XML to XML. E.g. BPEL offers such a mechanism through the *assign* tag. After the process execution model is finished, the modeler can derive orchestration or choreography languages from the model. The languages most known for the orchestration and choreography of services are BPEL and WS-CDL. An MDA approach in the field of SOA has already been presented in [19].

Derivation of service orchestrations: BPEL describes the business process from a particular partner's point of view. In most cases, however, the service calls will cross enterprise boundaries and involve other entities such as subsidiaries or subcontractors. If each business partner describes the business process from his own point of view, the final process specifications will most likely not match. By using a more holistic modeling approach such as UMM (UN/CEFACT's Modeling Methodology) [14] the modeler can describe the inter-organizational business process in a semantically unambiguous way. A UMM model which is based on UML [12] can then be used to generate artifacts for a service oriented environment. We have already shown the feasibility of such an approach with BPEL in [6] which is based on research conducted in [4]. The transformation shown in [6] is limited to abstract processes in BPEL because UMM currently does not incorporate service bindings. Nevertheless with service bindings implemented, UMM promises to be a valuable design methodology for applications in a service oriented context.

It will heavily depend on the availability of appropriate tools, whether composite applications are successful or not. The development of composite applications is the second crucial issue after the infrastructural question, an enterprise is facing when introducing a service oriented architecture. Due to the importance of the applications used in a service oriented context, a potential SOA roadmap must incorporate a detailed strategy for the software to be used.

2.3. Innovation and standards

Whether the introduction of a service oriented architecture has an impact on innovation policies or not depends on several factors. If the enterprise is rather small, the introduction of a service orientation will most likely be outsourced and hence no real innovation takes place. For medium and large-sized enterprises the introduction of SOA increases the potential for innovation. Especially when the architecture is developed in-house and not outsourced the SOA approach can be regarded as an innovation promoter. Due to the joint effort all departments of the enterprise have

to make, new processes and techniques are developed ultimately leading to a new architecture - namely the service oriented architecture.

Most of the client/server systems are self-implemented solutions. The necessary programming languages and standards are well known to most of the IT staff today. Self-implemented software is often written on the basis of an enterprise server such as JBoss or similar products. The standards and techniques necessary to extend the software, are well known. With the rise of service oriented architectures however, a whole new set of technologies and methods starts to dominate the IT sector. Apart from the core XML-based standards already mentioned a whole new set of WS-* standards has been developed such as WS-Interoperability (WS-I), WS-Security (WS-S), Security Services (SAML) and Web Services Reliable Messaging (WSRM) just to name a few. With the increased proliferation of a service orientation such standards and their correct use become an important issue in IT departments. Although important, the current knowledge of IT staff in regard to these standards is quite low. One crucial part of a SOA roadmap must be the identification of the standards necessary. In the next step the knowledge level of the standards within the enterprise must be elicitated and necessary steps must be taken in order to overcome any knowledge gaps.

2.4. Sourcing

The effective implementation of a service oriented architecture very much depends on the capabilities of the IT department. In order to guarantee a seamless transition from the status quo to a service oriented environment, knowledge about service integration, composition and life-cycle management within the IT department is necessary. Especially in small and medium sized businesses such a knowledge is often not present. Whether a service oriented architecture is realized or not therefore often relies on a *make* or *buy* decision.

The make decision: When talking about a *make* decision the IT responsible refers to the in-house creation of specific software. In regard to service oriented architectures the advantage of a *make* decision becomes clear. The enterprise can choose the technology used and contribute with the internal know-how to the creation of the software. Furthermore, the integration of existing technologies and partners such as suppliers, subsidiaries and subcontractors is possible. The architecture created is tailored to the specific needs of the business and there is no dependence on external partners in regard to software maintenance and use. After the IT department has successfully carried out the in-house implementation of and transition to SOA a valuable amount of knowledge is available for further projects and potential for future synergy effects is given. On the other hand the *make* approach is cost and time-intensive and contains a certain amount of risk which must not be underestimated. In order to pursue a *make* decision specific domain knowledge, especially in the field of service engineering, orchestration and implementation is needed. If the knowledge is not available additional staff must be hired or existing staff be trained. The fact that such knowledge or staff is often not present leads us to the *buy* decision.

The buy decision: If the know-how of the IT department is not sufficient to implement a holistic service oriented architecture internally, an external partner and external software must be engaged respectively. Another important factor for the external development and implementation of a service oriented architecture is the lack of resources in the enterprise. Hiring new qualified employees just for the implementation of the service orientation contains a high financial risk whereas outsourcing the task minimizes the financial and the failure risk. The engagement of an external partner for the realization of a service oriented architecture is often cheaper and faster than the in-house realization. Although the planning and implementation of the service orientation can be outsourced, an enterprise should train in-house staff for the maintenance of the systems. As already mentioned before a service oriented context is volatile - services are abandoned or assembled to new services often within relatively short intervals. If necessary staff for service adaptation or correction is available in-house and the enterprise can react faster to new business needs.

2.5. Investment

A traditional investment, regardless if undertaken by department *x* or the IT department in most cases has one aim: a positive return on investment (ROI). Apart from being positive the investor expects the ROI to be quick. Considering the investments necessary for a service oriented architecture we can identify three different types according to [7]: organizational, architectural and infrastructural investments. Investments from the organizational perspective include putting in place new and specific processes such as service set-up or service deployment as well as human resources. The architectural investments include the evaluation, planning and testing of new software architectures. On the infrastructural side the investments mainly affect the software artifacts and the hardware necessary for a service orientation. All three investment types have one financial aim in common with the service oriented approach - the lowering of process costs. However, the initial investments for establishing a SOA architecture are higher than pursuing and maintaining a traditional architecture. A SOA project does not produce a quick ROI - hence it must not be induced on a ROI opportunistic basis. In the long-term however, the SOA approach produces a positive ROI due to in-

creased competitiveness, faster reaction on market changes and needs and lowered process costs. Furthermore the total cost of ownership of software is decreasing and the time-to-market of new products and services is lowered.

2.6. Human resources

The human resources sector will encounter significant changes with the introduction of a service oriented environment. In a classic client/server environment the development of new systems and applications was commissioned by the business executives to the responsible project managers. Project managers hired programmers who coded the applications according to the requirements elicitated. At the time the development of the application was finished and it was released the initial business requirements had often already changed. Adaptations to changes and extensions to the software were complicated and time and cost intensive. This classic view on the roles in software engineering must not be regarded as outdated - however, in a service oriented world a new paradigm prevails. Figure 5 gives

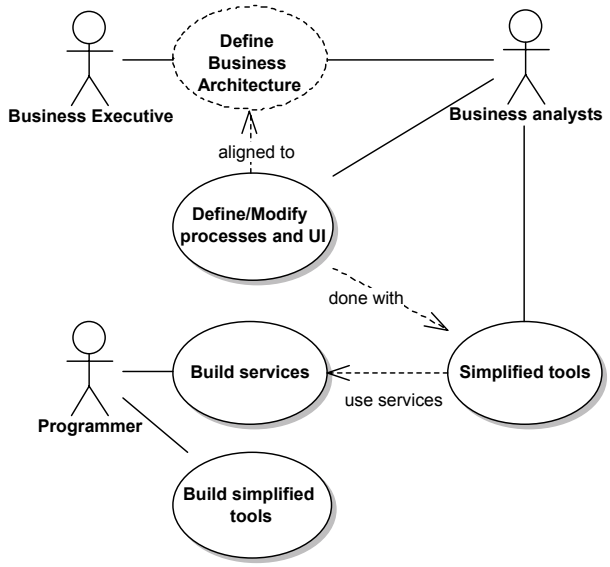


Figure 5. Roles in a service oriented environment

an abstract overview about the roles and the relevant use cases in a service oriented world. A similar approach has already been scrutinized in [2]. The business architecture is defined by the business executives and the business analysts. Because of the application of model driven design tools the business analysts can define and modify processes and user interfaces. The help of programmers will still be needed for special cases and exceptional behavior, but generally the business analysts can lower the workload of programmers. The tools used by the business analyst are developed by programmers who incorporate the services they have build into the tools. This approach greatly enhances the speed of application development. It however, implies that business analysts are available and have the knowledge about the relevant domain. Therefore a major task in adopting SOA within an enterprise is the hiring of business analysts with adequate know-how or the training of existing employees.

For the programmers within an enterprise it is important to get acquainted to the new technologies required for a service orientation. The main technology areas of interest are the different XML-based standards in use and the internal architecture necessary for service orientation. Especially when setting up a service orientation on top of existing systems, a wrapping of old business logic is necessary. I.e. the mainframe system used for accounting could be wrapped with a web service in order to make its functionality available to subsidiaries. Building such wrappers requires either the implementation directly in the legacy system - in this case on the mainframe - or writing an intermediate layer performing the conversion from a system call to an XML message. In both cases profound knowledge of the legacy system is required. However, such legacy systems are often in use for 20 years or longer and the programmers often have already retired. One major task in making legacy systems available in a service oriented world will be the hiring of qualified programmers who can implement the necessary interfaces.

In a roadmap to a service oriented environment the human resources factor plays a very important role. Without qualified employees a transition to SOA will most likely fail or at least be inefficient.

3. Conclusion and Outlook

The current strong trend towards service orientation is unlikely to change in the next few years. Even more than service orientation supporting techniques such as model driven architectures and model driven development will evolve from their draft status into fully qualified software development tools. With the availability of such tools the proliferation of service oriented concepts will increase significantly. Moreover, with the availability of service oriented commercial of the shelve software (COTS) also small and medium sized businesses can participate on a service oriented level. The augmentation of service orientation together with supporting tools will help composite applications to be as widespread as client/server applications are nowadays. Therefore it is more crucial than ever for an enterprise to stay up to date and not to miss the service orientation train. Whether there is business conducted between two arbitrary companies or not will very much rely on the

fact, if the other partner supports a service oriented environment. It has already been shown how many different sections of an enterprise are affected by the introduction of a service oriented environment. Moreover, it is most likely easier to name those departments not affected by the introduction than those affected. Hence the design and implementation must be thoroughly planned and organized.

On top of the organization the overall SOA strategy is placed, guiding the enterprises' efforts towards a functioning service oriented architecture. In order to fulfill the requirements and goals set by the strategy, a SOA roadmap is needed. In the long term only those enterprises will be successful which have a well developed and adapted SOA roadmap.

This work was supported by the Austrian Federal Ministry of Economics and Labour and the IST Sixth Framework Programme of the European Union.

References

[1] BEA, IBM, Microsoft, SAP AG and Siebel Systems. *Business Process Execution Language for Web Services*, May 2003. Version 1.1.

[2] Dan Woods, Thomas Mattern. *Enterprise SOA, Designing IT for Business Innovation*. O'Reilly, 2006.

[3] Frank E. Redmond III. *Microsoft Distributed Component Object Model*. John Wiley and Sons Inc, 1997.

[4] B. Hofreiter and C. Huemer. Transforming UMM Business Collaboration Models to BPEL. In *Proceedings of OTM Workshops 2004*, volume 3292, pages 507–519. Springer LNCS, 2004.

[5] Ivar Jorstad, Schahram Dustdar, Do Van Thanh. A service oriented architecture framework for collaborative services. In *Proceedings of the 14th IEEE International Workshops on Enabling Technologies: Infrastructure for Collaborative Enterprise (WETICE05)*, pages 121–125. IEEE, 2005.

[6] P. Liegl, R. Schuster, and M. Zapletal. A UML Profile and Add-In for UN/CEFACT's Modeling Methodology. Master's thesis, University of Vienna, February 2006.

[7] Massimo Pezzini. Gartner Group Applied SOA Presentation: Best Practices from the Best Practitioners.

[8] Michael E. Porter. *Competitive Advantage*. Simon and Schuster, January 2004.

[9] Michael J. Earl. *Information Management: The Organizational Dimension*. Clarendon Press, March 1996.

[10] OASIS. *UDDI Version 2.04 API Specification*, July 2002. UDDI Committee Specification.

[11] Object Management Group (OMG). *Common Object Request Broker Architecture*, July 2002. Version 3.0.

[12] Object Management Group (OMG). *Unified Modeling Language Specification*, April 2005. Version 1.4.2.

[13] I. Sun Microsystems. Assessing your SOA readiness. Technical report, June 2004.

[14] UN/CEFACT Techniques and Methodologies Group. *UN/CEFACT's Modeling Methodology (UMM), UMM Meta Model - Foundation Module*, Oct. 2006. Technical Specification.

[15] Ward and Peppard. *Strategic Planning for Information Systems*. Academic Internet Publishers Incorporated, October 2006.

[16] World Wide Web Consortium (W3C). *Web Services Description Language (WSDL)*, Mar. 2001. Version 1.1.

[17] World Wide Web Consortium (W3C). *Simple Object Access Protocol (SOAP)*, June 2003. Version 1.2.

[18] World Wide Web Consortium (W3C). *Web Services Choreography Description Language*, Nov. 2005. Version 1.0, http://www.w3.org/TR/ws-cdl-10/.

[19] W. H. J. Y. Yanbing Jiang, Chunxiao Xing. On procedure strategy of constructing SOA's modeling language. In *Proceedings of the 2005 IEEE International Workshop on Service-Oriented System Engineering (SOSE05)*, pages 71–76. IEEE, 2005.

A Statistical Approach to Model-Based Robustness Testing

By
Miroslav Popovic and Jelena Kovacevic
University of Novi Sad, Department for Computing & Automation,
Trg D. Obradovica 6, 21000 Novi Sad, Serbia,
tel:+381-21-58-165, fax:+381-21-350-610, e-mail: rt_pop@krt.neobee.net

Abstract

Recent massive research in the area of robustness testing of communication protocols have been driven by the fact that embedded systems are becoming more and more complex every day. Providing correct implementations of communication protocols is a key to successful interconnection of embedded system constituents. Test and verification is the last and the most critical phase in the production of the corresponding software artifacts. This paper contributes to the overall research effort in the area by proposing an original approach to model-based robustness testing of communication protocols. In the paper we introduce the notion of hidden FSM states and state transitions. We also show how to calculate the probabilities of detecting the existence of the hidden state transitions. Then we present our methodology of model-based robustness testing of communication protocols. The methodology is based on the special operational profiles, which are referred to as the stress operational profiles.

1. Introduction

The communication protocol engineering [1] is an engineering discipline that deals with the engineering of communication protocols. Although frequently addressed in the literature, the methodology that governs the production process of the corresponding software artifacts remains work in progress. Particularly, the test & verification phase of the process is still the Achilles heel of the communication protocol engineering. UML 2.0 has improved some of these short-comes, but there is still room for further improvements in that area. For example, there are a number of formal verification methods (e.g. [3]) and statistical usage testing methods [4-7] at our disposal, which may be used to improve the current practice.

In our previous work [8-11] we have used our original approach to the model-based statistical usage testing of a class of communication protocol implementations that are based on the State design pattern [12] and Java programming environment augmented with the class *FSMSystem*. Most recently [11] we have created a novel model-based statistical usage testing working environment by improving the tools that we have developed previously [8-10] and by combining them with the unit testing framework JUnit and new tools that glued them together. The operational profiles that we used up to now modeled only the protocol reactions to the messages without faults.

Automatic testing of protocol implementation robustness was the next natural evolutionary step in our work. The IEEE defines *robustness* as a degree to which a system or component can function correctly in the presence of invalid inputs or stressful environmental conditions [13]. There is a lot of recent research in this area [14-17].

ASPIRE (Automated Systematic Protocol Implementation Robustness Evaluation) [14] is an automated approach to pro-actively test protocol implementations by observing their responses to faulty PDUs (Protocol Data Units) or messages. A PDU is faulty if either its syntax is incorrect or its position in a PDU sequence is incorrect (syntactically and semantically faulty PDUs). This approach uses the pairwise testing strategy [15] for generating syntactically faulty PDUs in order to limit the natural exponential growth in the number of the syntactically faulty PDUs. In this paper we use a special operational profile to generate both syntactically and semantically faulty PDUs and messages. We refer to such an operational profile as the *stress operational profile*.

In [16] the authors have extended the results of PROTOS project in multiple ways to overcome its limitations. They developed a test engine with callback functionality and dynamic option reloading feature that allows test engineer to have full control of protocol

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

state transition. They also contributed a new PDU generator that accepts test case templates written in a descriptive language similar to augmented BNF (RFC 2234). The new language is referred to as SBNF (Strengthened BNF). Its predefined and customizable functions are very similar to our intrinsic functions (which we use to specify possible values of event class parameters). However, they do not state anything about the quality of their test cases. In contrast to that paper, we calculate the mean significance level for the set of automatically generated test cases as an indicator of their quality.

VirtualWire [17] is a network fault injection and analysis system designed to facilitate the process of testing network protocol implementations. It allows developers to specify which network faults should happen when using event-based scripting language, and injects these faults into actual protocol test runs according to high-level specifications. Each scenario in the FSL (Fault Specification Language) is an unordered set of rules, which are {*condition* >> *action*} pairs. Actually, these scenarios are similar to our event class definition, but without their respective probabilities. Possibility of generating test scripts directly from the protocol specification is indicated as a direction for their future work. We already achieved that in our previous work [11].

In this paper we present our original approach to the model-based robustness testing of a class of protocol implementations, which are based on the State design pattern and Java programming environment. The approach is based on the application of the statistical usage testing. Essentially, we construct stress operational profiles, which model the behavior of the implementation under test when it is exposed to both valid and invalid messages. The *invalid messages* are either the syntactically faulty messages or the semantically faulty messages.

Once we create a stress operational profile, we use the tools that we developed in our previous work [8-11] and slightly improved in this paper, to automatically generate the test suite, check its quality by checking the mean significance confidence level, and automatically execute all the test cases and check their outcomes. We made the slight improvement of the working environment by introducing the class *FsmTestCase*.

The rest of the paper is organized as follows. We introduce the notion of hidden FSM (Finite State Machine) states and hidden FSM state transitions in Section 2. After that introduction we show the method for calculating probabilities of detecting the existence of the hidden FSM state transitions on the example of the SIP INVITE client transaction protocol. The methodology for the model-based robustness testing of communication protocols is presented in Section 3. The case study is given in Sections 4 and 5. Sections 6 and 7 contain closing remarks and references cited in the paper, respectively.

2. Hidden FSM states and transitions

FSM robustness evaluation is closely related to the problem of hidden FSM state transitions and states. The goal of the robustness tester is to find a sequence of syntactically and/or semantically faulty messages that will cause the FSM to take a hidden state transition to some of its regular states or to some hidden state, which will cause FSM's misbehavior later on during its evolution. FSM states and transitions are *regular* if they are defined in the FSM's original specification. Otherwise they are *hidden*.

Consider a simple example shown in Figure 1. The FSM comprises three regular states, namely *S0*, *S1*, and *S2*, as well as three regular state transitions, see Fig. 1.

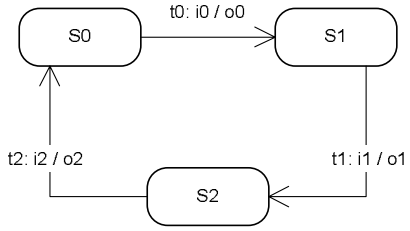


Figure 1. An example of a simple FSM

Let us assume that the FSM shown in Figure 1 was erroneously implemented with one hidden state (*S3*) and three hidden state transitions (*t3*, *t4*, and *t5*), see Fig. 2.

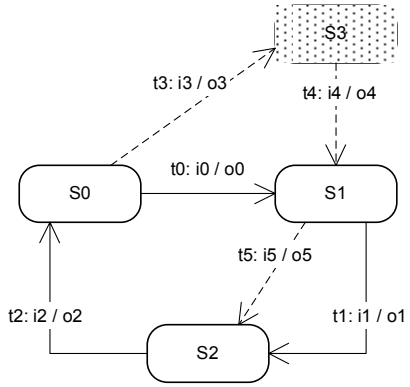


Figure 2. A FSM with implementation errors

If we assume that only outputs from the FSM are observable, it is clear that by testing the erroneous implementation only with the regular sequences of regular inputs (e.g. *i0*, *i1*, and *i2*) we can not discover hidden states (e.g. *S3*) and state transitions (e.g. *t3*, *t4*,

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

and *t5*). This means that if we want to discover them we must probe the FSM with the faulty inputs.

An input is faulty if either its syntax is incorrect or its position in an input sequence is incorrect. The former is referred to as a *syntactically faulty* input, whereas the latter is referred to as a *semantically faulty* input. For example, if *i3* is different from *i0*, *i1*, and *i2*, it is a syntactically faulty input (see Figure 2). Alternatively, if *i3* is equal either to *i1* or *i2*, it is a semantically faulty input.

The hidden state transitions can be discovered easily if they generate irregular outputs. An output is an irregular output if it does not appear in the original FSM specification (in the example in Figure 2 that would be any output different from *o0*, *o1*, and *o2*). Discovering hidden state transitions and hidden states may be more difficult if the hidden state transitions are silent or if they produce some of the regular outputs. A state transition is a *silent state transition* if it does not produce any output (we denote the absence of the output in the state transition label with the character -).

Certain hidden states and transitions that are silent, or that produce regular outputs, can not be discovered at all. For example, if the hidden transition *t3* in Figure 2 is a silent one and the hidden transition *t4* is equal to *t0* (this means that *i5* is equal to *i0* and *o5* is equal to *o0*), these two hidden state transitions and the hidden state S3 can not be discovered at all. However, this particular case is completely harmless since the erroneous FSM implementation has the same external behavior as the original FSM specification.

Alternatively, some of the hidden state transitions and states are discovered with certain sequences of inputs, but they remain undiscovered with other sequences of inputs. According to our experience from dealing with some real communication protocols, e.g. SIP, the probability of discovering hidden state transitions is rather high. As an example, consider the SIP INVITE client transaction, a part of the SIP protocol stack (refer to RFC 3261).

We restrict our analysis in this example to semantically faulty FSM inputs (messages) and to silent hidden state transitions only. It will be shown later in Sections 3 and 4 that stress operational profiles do not depend on particular hidden states and transitions. Let us introduce the abbreviated names for the FSM states, inputs, and outputs, which are shown in Table 1.

By introducing the abbreviations from Table 1 we obtain FSM statechart in Figure 3. Next we introduce the hidden state transitions. Then we assume that the hidden state transition has been taken and we analyze the output and the next state for each possible FSM input (*i0-i7*). Based on this analysis we calculate the probabilities of discovering hidden state transitions for individual FSM inputs. Once we determine these values, we calculate the probabilities of discovering individual hidden state transitions.

Table 1. The table of abbreviated names

No	Original name (from the RFC 3261)	Abbreviated name
States		
1	Calling	S0
2	Proceeding	S1
3	Completed	S2
4	Terminated	S3
Inputs		
1	INVITE	i0
2	TA fires	i1
3	300-699	i2
4	1xx	i3
5	TD fires	i4
6	TB fires	i5
7	TERR	i6
8	2xx	i7
Outputs		
1	INVITE sent	o0
2	ACK sent	o1
3	resp to TU	o2
4	1xx to TU	o3
5	Inform TU	o4
6	2xx to TU	o5

The probability of discovering the hidden state transition *ti*, denoted by p_i, is calculated as the mean value of all the probabilities of discovering the hidden state transition for the individual FSM inputs, denoted by p_{ij}, i.e.:

$$p_i = (p_{i1} + p_{i2} + \ldots + p_{in})/n$$

where:

n is the number of regular FSM inputs

p_{ij}, $j = 1, 2 \ldots n$ are the probabilities for the individual inputs.

Finally, we calculate the mean probability of discovering hidden state transitions, denoted by *p*, as follows:

$$p = (p_1 + p_2 + \ldots + p_m)/m$$

where:

m is the number of analyzed hidden state transitions

p_j, $j = 1, 2 \ldots m$ are probabilities for individual transitions.

The number of analyzed hidden state transitions in this particular example is the sum of the numbers of hidden state transitions that are starting from any state, except the state *S3* (*Terminated*), and going to all other states. The state *S3* is a special state for the SIP INVITE client transaction, because once this state is reached, the FSM is immediately destroyed. This means that state transitions from *S3* to *S0*, *S1*, or *S2* are not possible. Therefore, by excluding *S3* as a possible source of a hidden state transition, we get nine hidden state transitions that are subjects to further analysis.

We start the analysis with the hidden state transitions whose source is the state S0. These are the transitions S0-S1, S0-S2, and S0-S3. Generally, a state transition from the state Si to the state Sj is denoted as Si-Sj. These particular hidden state transitions are shown in Figure 4.

Table 2. The outcome analysis for Figure 4

Input\Trans.	S0-S1	S0-S2	S0-S3
i0	+-	+-	+-
i1	-	-	-
i2	++	-	-
i3	++	-	-
i4	+-	+-	+-
i5	-	-	-
i6	-	++	-
i7	++	-	-
Score G : B	5 : 3	3 : 5	2 : 6

The consequences of the given hidden state transition are determined by assuming that it has been taken and by analyzing the result of the next regular FSM input (see Table 2). A regular FSM input is an input that appears in the original FSM specification. We assume that all the inputs have the same occurrence probabilities. For brevity, we introduce the following notation:

- - denotes an unexpected (bad) output. If the result is -, an error (the existence of the hidden state transition) is discovered.
- ++ denotes the expected (good) output and the expected next state. If the result is ++, the error is not discovered and it will not be discovered by subsequent inputs.
- +- denotes the expected (good) output and the unexpected next state. If the result is +-, the error is not discovered, but it may be discovered by a subsequent input.

The last raw in Table 2 shows the score of expected versus unexpected FSM outputs (G : B stands for good against bad FSM outputs).

Next we analyze the hidden state transitions whose source is the state S1. Finally, we analyze the hidden state transitions whose source is the state S2. The summary of that analysis is given in Table 3. The probabilities of discovering particular hidden state transitions for given FSM inputs, p_{ij}, as well as the probabilities of discovering particular hidden state transitions, p_i, are given in Table 4.

The probability of discovering the hidden state transition *S2-S3* has the lowest value (41%). This makes sense because the FSM in the state *S2* reacts only to 3 out of 8 regular inputs (see Figure 3). Moreover, the regular state transition from the state S2 triggered by the input *i4* is a silent one. This means that only for 2 out of 8 regular inputs (these are *i2* and *i6*) the hidden state transition is discovered immediately. We may conclude that in this particular case the probability of discovering the hidden state transition is low because *the source state of the hidden state transition is the source of a relatively small number of regular state transitions and some of them are silent.*

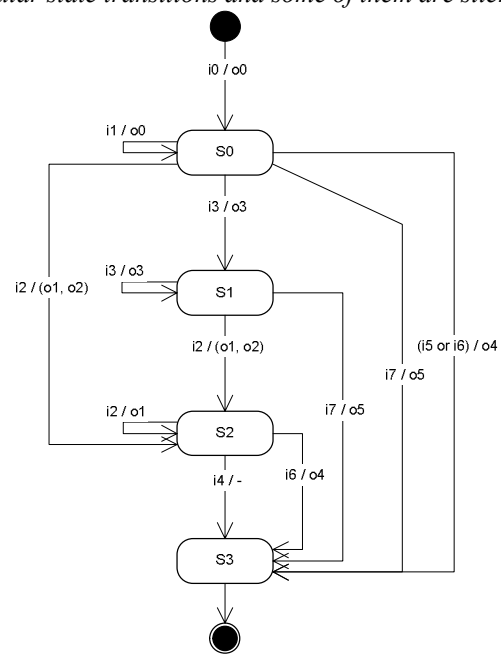


Figure 3. The FSM with the abbreviated names

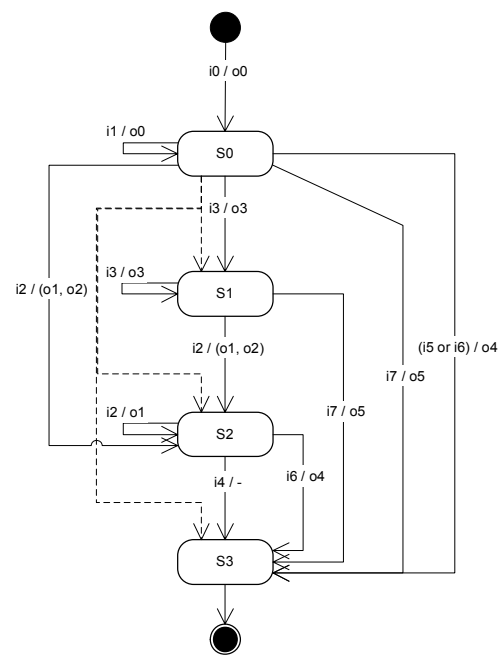


Figure 4. The FSM with hidden transitions from S0

Next lowest probability is the probability of discovering the hidden state transition *S0-S1* (42%). The situation that we have here is a very interesting

one. The FSM in the state *S1* also reacts to 3 out of 8 regular inputs (these are *i2*, *i3*, and *i7*), but these reactions are identical to the reactions in the state *S0*, which is the source state of the hidden state transition under examination. Both the generated output and the destination states are the same. The outcome in all three cases is ++, which means expected (good) output and the expected destination state. Consequently, the hidden state transition S0-S1 is discovered immediately only by other regular inputs that cause loud state transitions from the state *S0* (these are *i1*, *i5*, and *i6*). The state transition is a *loud state transition* if it is not a silent one.

Table 3. The summary of the outcome analysis

Trans.	i0	i1	i2	i3	i4	i5	i6	i7	G:B
S0-S1	+-	-	++	++	+-	-	-	++	5:3
S0-S2	+-	-	-	-	+-	-	++	-	3:5
S0-S3	+-	-	-	-	+-	-	-	-	2:6
S1-S0	+-	-	++	++	-	-	-	++	4:4
S1-S2	+-	+-	-	-	-	+-	-	-	3:5
S1-S3	+-	+-	-	-	+-	+-	+-	-	5:3
S2-S0	+-	-	-	-	+-	-	++	-	3:5
S2-S1	+-	+-	-	-	+-	+-	-	-	4:4
S2-S3	+-	+-	-	+-	+-	+-	-	+-	6:2

Table 4. The probabilities of discovering hidden trans.

Trans.	i0	i1	i2	i3	i4	i5	i6	i7	p_i
S0-S1	0	1	0	0	3/8	1	1	0	.42
S0-S2	0	1	1	1	6/8	1	0	1	.72
S0-S3	0	1	1	1	6/8	1	1	1	.84
S1-S0	0	1	0	0	1	1	1	0	.50
S1-S2	0	5/8	1	1	1	5/8	1	1	.78
S1-S3	0	3/8	1	1	3/8	3/8	3/8	1	.56
S2-S0	0	1	1	1	5/8	1	0	1	.70
S2-S1	0	4/8	1	1	4/8	4/8	1	1	.69
S2-S3	0	2/8	1	2/8	2/8	2/8	1	2/8	.41
								p =	.63

We may conclude that in this particular case the probability of discovering the hidden state transition is low because *the FSM loud reactions in the destination state of the hidden state transition are the subset of the FSM loud reactions in the source state of the hidden state transition*. This conclusion is based on the assumption that two FSM reactions are identical if the FSM produces the same outputs and finishes in the same destination state (the source states may be different).

The highest probability is the probability of discovering the hidden state transition *S0-S3* (84%). This result also makes sense because the FSM in the state S0 reacts loudly to 6 out of 8 regular FSM inputs (these are the inputs *i1*, *i2*, *i3*, *i5*, *i6*, and *i7*) and silently to 2 out of 8 regular FSM inputs (these are the inputs *i0* and *i4*), whereas the FSM in the state *S3* reacts silently to all the regular FSM inputs. This means that in 6 cases out of 8 the existence of the hidden state transition is discovered immediately and that is why the corresponding probability is high. We may conclude that in this particular case the probability of discovering the hidden state transition is high because *the FSM reactions in a source state of the hidden state transition are loud for most of the FSM regular inputs, whereas the FSM reactions in the destination state of the hidden state transition are silent*.

Based on the previous analysis it may seem that the probability of discovering a hidden state transition is proportional to the number of differences for the FSM reactions in the source and destination states of the hidden state transition. Without giving it a second though, we could naively conclude that the larger the number of reaction differences (sum of - and +- outcomes), the highest the probability of discovering the hidden state transition. However, when we construct the table that shows the relation between these two variables (see Table 5) we see that this conclusion is wrong.

Table 5. Probability vs. reaction differences

Transaction	Differences (- and +-)	p_i
S0-S1	5	0.42
S0-S2	7	0.72
S0-S3	8	0.84
S1-S0	5	0.50
S1-S2	8	0.78
S1-S3	8	0.56
S2-S0	7	0.70
S2-S1	8	0.69
S2-S3	8	0.41

By analyzing Table 5 we see that there are two probability values (42% and 50%) that correspond to the reaction differences value 5. Since these two values are rather close, we could proceed by calculating the average value (46% to be exact), but the problem that we face is the fact that for the reaction differences value 8 there are 5 different probability values in the range from 41% to 84% (these values are 41%, 56%, 59%, 78%, and 84%).

The reason for such results is that we are observing only the FSM outputs, whereas the FSM current state remains its private matter. That fact leads us to the assumption that the number of the FSM output differences (- outcomes) and the probability of discovering a hidden state transition are more closely related.

By constructing Table 6 we provide encouraging results for this direction of reasoning. Finally, by calculating the average probabilities for each value of the FSM output differences (see Table 7) we show that *the average probability of discovering hidden states with the same FSM output differences value is*

proportional to that value. The results from Table 7 are illustrated in Figure 5.

Table 6. Probability vs. output differences

Transaction	Differences (-)	p_i
S0-S1	3	0.42
S0-S2	5	0.72
S0-S3	6	0.84
S1-S0	4	0.50
S1-S2	5	0.78
S1-S3	3	0.56
S2-S0	5	0.70
S2-S1	4	0.69
S2-S3	2	0.41

Table 7. Average probability vs. output differences

Transaction	Differences (-)	p_a
S2-S3	2	0.41
S0-S1, S1-S3	3	0.49
S1-S0, S2-S1	4	0.60
S0-S2, S1-S2, S2-S0	5	0.73
S0-S3	6	0.84

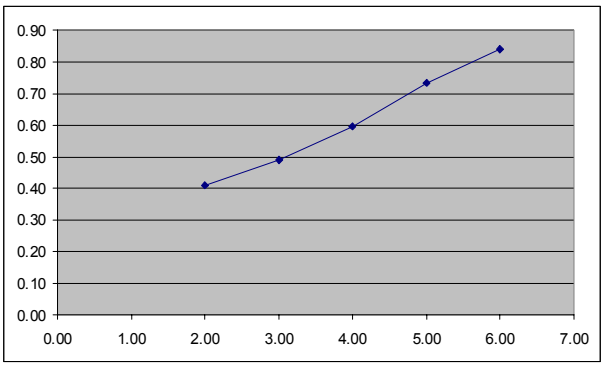


Figure 5. Average probability vs. output differences

3. Methodology

This section of the paper outlines the methodology for the robustness testing of the class of communication protocol implementations. The methodology consists of the following steps:

(1) Make a model of the stress operational profile.
(2) Interpret the model.
(3) Calculate the number of required test cases based on the desired level of product reliability.
(4) Generate generic test cases.
(5) Check the statistical report.
(6) If the mean significance confidence level is not higher than 20%, return to step (4) above.
(7) Compile generic test cases into JUnit test cases.
(8) Execute JUnit test cases.
(9) Report any discrepancies to the design and implementation team.

Stress operational profile models the implementation under test behavior when the implementation is exposed to both valid and invalid inputs (messages). Essentially, this model is a Markov process, which is characterized by the given probabilities of individual state transitions. The goal of the stress operational profile is to create the stressful usage by extending the normal usage patterns that are driven by valid messages with the unexpected usage patterns that are driven by both syntactically and semantically faulty messages.

If we want to create more stressful environment, we increase the probabilities of invalid messages. If we want to create less stressful environment, we increase the probabilities of valid messages. Selecting proper probabilities is a critical task. Very large probabilities of invalid messages may slow down normal protocol evolution significantly. The probability value of 100% blocks the protocol evolution completely. At the moment we use heuristics to set the probabilities of invalid messages.

A *model interpreter* transforms the above mentioned model into the operational profile specification file. This file specifies the number of states and event classes, as well as the state transition probabilities, the event class definitions, and the next state definitions for each state transition in the model. It is used in the step 4.

The required number of test cases (N) for the desired level of product reliability (R) and the upper bound on the probability that model assertions are in error (M) is calculated from the following formula [4]:

$$M = R^N$$

A test case consists of a given number of test steps. This number is referred to as the *test case length*. A *test step* actually corresponds to the state transition. An automatic test case generator generates a test case by a random walk through the stress operational profile. It starts from the given initial state and makes the given number of random test steps in accordance with the given probability distribution. This procedure is repeated for the given number of test cases.

A particular test suite of the finite size has some deviation from the probability distribution specified by the stress operational profile. Statistically, the smaller the test suite size, the bigger the deviation. The main statistical measure of this deviation is the mean significance confidence level (SL). SL higher than 20% is considered large, whereas the SL smaller than or equal to 1% is considered small [4].

4. Case study

This section of the paper presents a case study – robustness testing of the SIP INVITE client transaction (IETF RFC 3261) implementation that is based on the class *FSMSystem* and the State design pattern. We start by modeling the stress operational profile which is used to test the robustness of the implementation under test. The model is created in accordance with the operational profile modeling paradigm, which is described in the previous sections of this paper.

The communication protocol under study is already described in Section 2 of this paper. Its statechart diagram is shown in Figure 3. The stress operational profile model is constructed from the statechart diagram by adding the reactions to the unexpected events (messages) and by adorning all the state transitions with their corresponding probabilities (see Figure 6).

The FSM ignores the unexpected messages in all its states. This means that the FSM remains in the same state and that it does not produce any output, i.e. it does not send any messages to other FSMs. That is why all the reactions to the unexpected messages have the label *Unexpected/-*. The exception to this rule is the state *Terminated*. Once the FSM reaches that state it is ultimately deleted and therefore it can not react to any message. Therefore the state transition from the state *Terminated* to the same state has the simple label *End/-* (the name *End* actually corresponds to the primitive *nop*). Of course, the corresponding probability must be 100%.

For simplicity, all the reactions to the unexpected messages are labeled *Unexpected/-* (see Figure 6). The name *Unexpected* that is used in this label actually refers to a set of syntactically and semantically faulty messages. Of course, these sets are different for various FSM states. For example, the semantically faulty messages in the state *Initial* are the messages *TA FIRES*, *300-699*, *1xx*, *2xx*, *TERR*, *TB FIRES*, and *TD FIRES*. A syntactically faulty message is any message that is not a regular message or a semantically faulty message. Such a message is referred to with the name *UNKNOWN*. All the reactions to the unexpected messages specify a syntactically faulty message as a single alternative to all the semantically faulty messages.

Another detail that distinguishes reactions to the unexpected messages in different states is the status of timers. Generally, the status of timers before and after these state transitions must remain the same (this is also implicitly assumed by the statechart diagrams). Obviously, the statuses of timers change as the FSM evolves from the current state to its next state, which is determined by the type of the driving stimuli.

The experiments in this case study were organized as follows. At the beginning, we generated five test suites with 1000, 2000, 3000, 4000, and 5000 test cases respectively. The aim was to see how the number of test cases affects the measurement results. The results that were the most interesting to us were the sequence number (starting from 0) of the first test case that discovers the existence of the hidden state transition and the total number of the test cases that detect the existence of the hidden state transition.

We planned in advance to do the actual experiments in the following three steps:

- Step 1: check the bug-free implementation against five test suites.
- Step 2: inject the hidden state transition *S0-S3* (*Calling-Terminated*) into the bug-free implementation and check the infected implementation against five test suites.
- Step 3: inject the hidden state transition *S2-S3* (*Completed-Terminated*) into the bug-free implementation and check the infected implementation against five test suites.

The test case length was selected to be the minimum number of steps that are needed to detect the hidden state transition *S2-S3*, which turned out to be 4 steps (step I: *INVITE*, step II: *300-699*, step III: the semantically faulty message *2xx*, and step IV: detection). Since there are many other test cases not reaching the state *S2* in the second test step, we expected relatively small percentage of test cases discovering the hidden state transition *S2-S3*.

Although the operational profiles described here and in Section 2 are rather different (the latter assumes uniform probability distribution and does not take into account syntactically faulty messages), it was reasonable to expect that the overall behavior of the protocol under test may be similar. Because of that, the hidden state transitions *S0-S3* and *S2-S3* were selected based on the findings made in Section 2, which indicated them as the hidden state transitions with the highest and the lowest probability of detecting them after they have been taken, respectively.

In accordance with the test plan outlined above, we conducted the step 1 first. All the test cases within the five test suites passed successfully. Thus the implementation under test turns out to be bug-free, or at least bug-free according to these five test suites (15.000 test cases all together, with four test steps, i.e. operational profile state transitions, each).

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

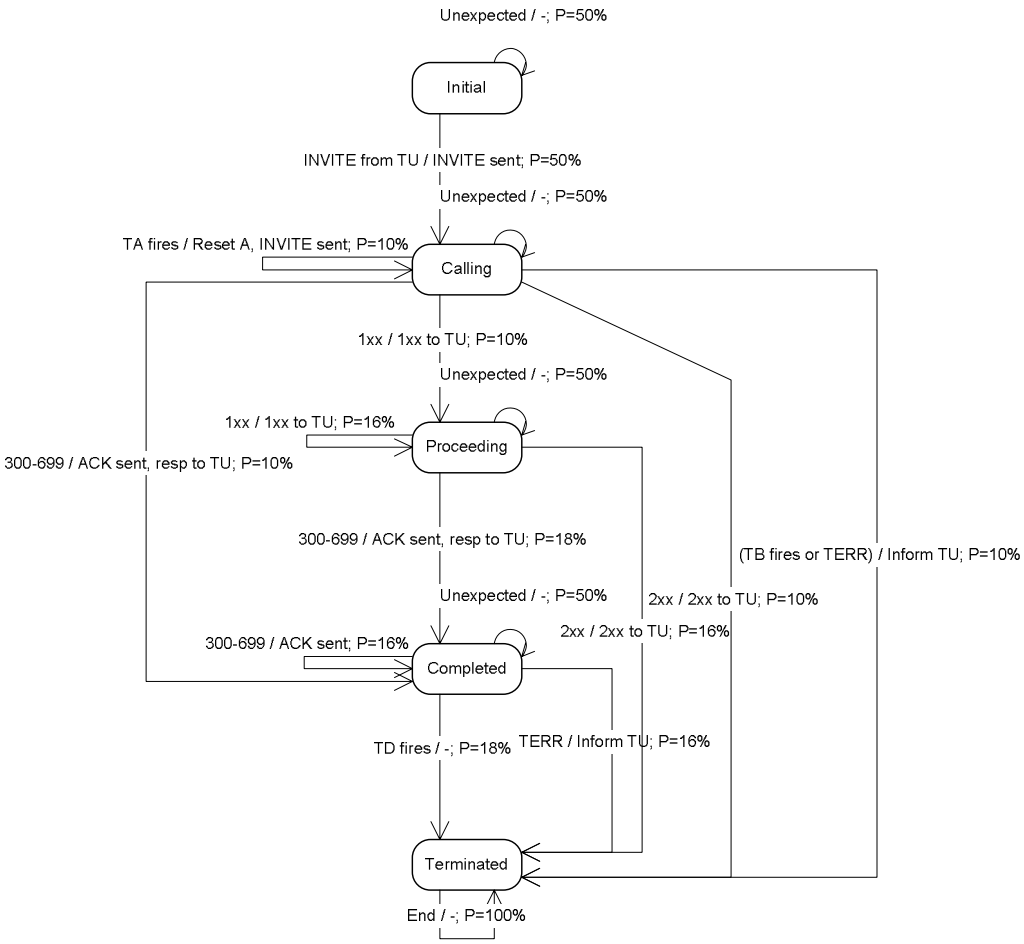


Figure 6. The stress operational profile model

Next we conducted step 2 by injecting the hidden state transition *S0-S3* (*Calling-Terminated*), which is triggered with the semantically faulty message *i4* (*TD FIRES*) and which does not perform any operation on any timer (because in Section 2 we assumed the detection of hidden state transitions based on the observation of the FSM outputs only). All five test suites detected the existence of the hidden state transition rather early (see Table 8) and with a considerable number of the total failed test cases (see Table 9).

Finally, we conducted step 3 by injecting the hidden state transition *S2-S3* (*Completed-Terminated*), which is triggered by the semantically faulty message *i7* (*2xx*) and which does not perform any operation on any timer. All five test suites detected the existence of the hidden state transition later than in the previous case (see Table 8) and with a rather small number of the total failed test cases (see Table 9).

The result of our experiments are summerized in Table 8 and Table 9. These results prove that it is really much easier to detect the existenece of the hidden state transition *S0-S3* than the hidden state transition *S2-S3*.

Table 8 is organized as follows. The rows of the table correspond to the individual test suites, namely the test suites with 1000, 2000, 3000, 4000, and 5000 test cases. The second and the third column show the test cases that were first to fail. The former shows those that were first to fail for the implementation under test infected with the hidden state transition *S0-S3*, whereas the latter shows those that were first to fail for the implementation infected with the hidden state transition *S2-S3*.

According to Table 8, the average test case sequence number that detects the existence of the hidden state transition *S0-S3* is 15, whereas the average test case sequence number that detects the existence of the hidden state transition *S2-S3* is 147. This leads to the interesting conclusion that on average we need almost 10 times more test steps to detect the hidden state transition *S2-S3* than we need to detect the hidden state transition *S0-S3*.

Table 9 is organized analogously to Table 8. The difference between the two is that Table 9 shows the total number of test cases that failed. The second column indicates the total number of test cases that failed for the implementation infected with the hidden state transition *S0-S3*, whereas the third column shows the results for the implementation infected with the hidden state transition *S2-S3*.

Table 8. The test cases that were first to fail

Test suite	H.S.T. S0-S3	H.S.T. S2-S3
1000	16	95
2000	24	183
3000	23	25
4000	8	33
5000	3	399
Average	15	147
Ratio	1	9.93

Note: H.S.T. stands for the Hidden State Transition.

Table 9. The total number of test cases that failed

Test suite	H.S.T. S0-S3	H.S.T. S2-S3
1000	90	2
2000	158	2
3000	220	8
4000	327	3
5000	386	9
Density [%]	7.87	0.16
Ratio	49,21	1

In addition to absolute total number of failures, we are interested in the failures *density*, which is defined as the ratio of the total number of failures and total number of test cases. Table 9 shows that the failures density is 7.87% for the implementation infected with the hidden state transition *S0-S3* and 0.16% for the implementation infected with the hidden state transition *S2-S3*. This leads to the interesting conclusion that the failure density is almost 50 times greater in the former case than in the latter. Next we conducted a series of experiments searching for means to improve the overall testing efficiency.

5. Improving the testing efficiency

In the second part of the case study we decided to further experiment with the minimal test effort of 4000 test steps (1000 cases x 4 steps) that was needed to reliably detect the hidden state transition *S2-S3*, based on the minimal test case length that was needed to detect it. We examined a series of test suites with different number of test cases and number of test case lengths, but with the overall test effort fixed to 4000 test steps. The results are given in the Table 10 and in the Figure 7.

By analyzing the case of the H.S.T. S0-S3 in the Table 10, we see that the number of broken test cases is proportional to the total number of test cases. Alternately, in the case of the H.S.T. S2-S3, the number of the broken test cases actually depends on the test case length, and has its maximum (value 7) for the test case length of 20 test steps.

Table 10. No. of broken cases vs. test case length

Test suite	SLi [%]	H.S.T. S0-S3	H.S.T. S2-S3
1000 x 4	26	86	1
800 x 5	48	96	1
400 x 10	21	90	4
267 x 15	41	67	4
200 x 20	36	46	7
160 x 25	58	40	5
134 x 30	49	34	5
114 x 35	60	28	5
100 x 40	68	36	3

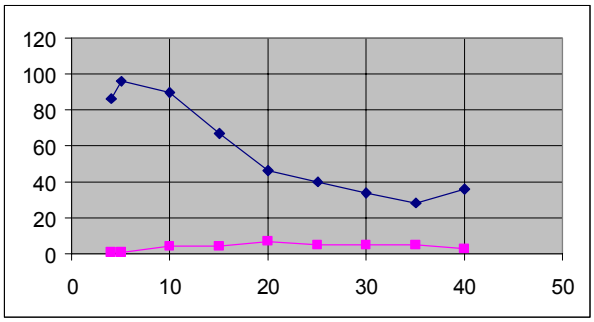


Figure 7. The no. of broken test cases vs. case length

These conclusions suggest that it should be possible to construct an effective test suite with much smaller overall test effort by fixing the test case length to 20 and reducing the total number of test cases. Of course, the final goal is to find the effective test suite with the minimal overall test effort. The results of the experiments that were made in that course are given in the Table 11 and in the Figure 8.

Table 11. No. of broken test cases versus test effort

Test suite	SLi [%]	H.S.T. S0-S3	H.S.T. S2-S3
200 x 20	36	46	7
100 x 20	40	24	9
20 x 20	48	7	2
10 x 20	33	5	0

By analyzing the results in the Table 11 (Figure 8) we see that an effective test suite of much smaller overall test effort exists, indeed. The particular test suite with only 20 test cases and 20 test steps per each test case has an order of magnitude smaller overall test effort of 400 test steps altogether.

6. Conclusions

In this paper we have introduced the notion of hidden FSM states and state transitions. By analysis in Section 2 we have shown that, although it may seem

logical, *the probability of detecting the hidden state transition is not proportional to the number of reaction differences (sum of - and +- outcomes) caused by taking the hidden transition*. By further analysis in Section 2 we have shown that *the average probability of discovering hidden states with the same FSM output differences value is proportional to that value*.

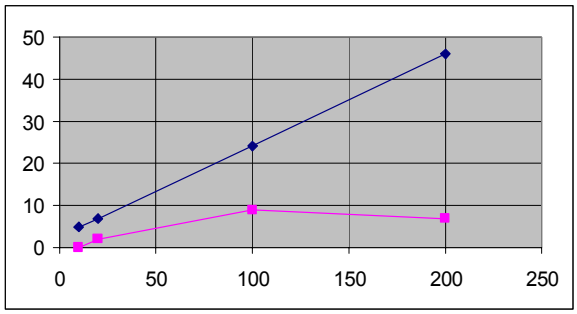


Figure 8. The no. of broken cases vs. no. of cases

Then we have presented our methodology of the model-based robustness testing of communication protocols. The methodology is based on the special operational profiles, which are referred to as the stress operational profiles. The goal of the *stress operational profile* is to create the stressful usage by extending the normal usage patterns that are driven by valid messages with the unexpected usage patterns that are driven by both syntactically and semantically faulty messages.

Finally, we have shown the application of the methodology on a case study. The results of the case study show that general conclusions mentioned above are true. Additionally, the case study shows that significant test effort may be needed for detecting hidden state transitions with the small number of FSM output differences (such as H.S.T. S2-S3) when test cases of minimal length are used. The test effort may be reduced for the order of magnitude by finding the optimal test case length and by reducing the number of test cases to the minimal number that is needed to experimentally construct the effective test suite, which reliably detects all the hidden state transitions.

7. References

[1] M. Popovic, Communication Protocol Engineering, CRC Press, Boca Raton, Florida, 2006, ISBN 0849398142.

[2] UML 2.0 Specifications, available for free download at http://www.omg.org/technology/documents/formal/uml.htm

[3] M. Popovic, V. Kovacevic, I. Velikic, "A Formal Software Verification Concept Based on Automated Theorem Proving and Reverse Engineering", Proceedings of 9th IEEE International Conference on the Engineering of Computer-Based Systems, Lund, Sweden, 8-11 April, 2002.

[4] D.M. Woit, "Operational Profile Specification, Test Case Generation, and Reliability Estimation for Modules", Ph.D. Thesis, Queen's University Kingstone, Ontario, Canada, February 1994.

[5] J.H. Poore, C.J. Trammell, "Application of Statistical Science to Testing and Evaluating Software Intensive Systems", in Science and Engineering for Software Development: A Recognition of Harlan D. Mills' Legacy, May 1999, pp. 40.

[6] M. Hubner, I. Philippow, M. Riebisch, „Statistical Usage Testing Based on UML", Proc. of 7th World Multiconferences on Systemics, Cybernetics and Informatics, July 27-30, 2003, Orlando, FL, USA. Vol. 1, IIIS, 2003. ISBN 980-6560-01-9. pp. 290-295.

[7] Y. Jiong, W. Ji, and C. Huowang, "Deriving Software Statistical Testing Model from UML Model", Proceedings of the Third International Conference On Quality Software (QSIC'03), Beijing, Sept 25-26, 2003.

[8] I. Velikic, M. Popovic, V. Kovacevic, "A Concept of an Integrated Development Environment for Reactive Systems", Proceedings of the 11th IEEE Conference on Engineering of Computer Based Systems, ECBS 2004, Brno, Czech Republic, May 24-26, 2004.

[9] M. Popovic, Z. Jurca, V. Kovacevic, "An Automatic Test Generator Engine", The 15th IEEE International Symposium on Software Reliability Engineering, industrial track, Saint-Malo, Bretagne, France, November 2-5, 2004.

[10] M. Popovic and I. Velikic, "A Generic Model-Based Test Case Generator", Proceedings of the 12th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems, Greenbelt, Maryland, Washington DC, USA, April 4-7, 2005.

[11] M. Popovic, I. Basicevic, I. Velikic, and J. Tatic, "A Model-Based Statistical Usage Testing of Communication Protocols", Proceedings of the 13th Annual IEEE International Conference and Workshop on Engineering of Computer Based Systems, Potsdam, Germany, March 27-30, 2006.

[12] E. Gamma et all, Design Patterns: Elements of Reusable Object-Oriented Software, Addison-Wesley, Upper Saddle River, NJ, 1995, ISBN 0-201-63361-2.

[13] I. S. 610.12-1990. IEEE Standard Glossary of Software Engineering Terminology. Dec. 1990.

[14] A. Vasan, A.M. Memon, "ASPIRE: Automated Systematic Protocol Implementation Robustness Evaluation", Proceedings of the 2004 Australian Software Engineering Conference (ASWEC'04), April 2004, pp. 241.

[15] K.C. Tai, Y. Lie, "A Test Generation Strategy for Pairwise Testing", IEEE Transactions on Software Engineering, January 2002, pp. 109-111.

[16] S. Xiao, L. Deng, S. Li, X. Wang, "Integrated TCP/IP Protocol Software Testing for Vulnerability Detection", Proceedings of the 2003 International Conference on Computer Networks and Mobile Computing (ICCNMC'03), October 2003, pp. 311.

[17] P. De, A. Neogi, T. Chiueh, "VirtualWire: A Fault Injection and Analysis Tool for Network Protocols", Proceedings of the 23rd IEEE International Conference on Distributed Computing Systems (ICDCS'03), May 2003, pp. 214.

Towards Model-Based Testing with Architecture Models

Stephan Schulz
European Telecommunications Standards Institute (ETSI)
F-06921 Sophia Antipolis Cedex
Stephan.Schulz@etsi.org

Jukka Honkola
Nokia Research Center
P.O. Box 407
FIN-00045 NOKIA GROUP
Jukka.Honkola@nokia.com

Antti Huima
Conformiq Software Ltd.
Tekniikantie 12
FIN-02150 Espoo
Antti.Huima@conformiq.com

Abstract

As tools are maturing rapidly model-based testing is gaining wider attention by industry as a potential replacement for suite-based testing approaches. To use model-based testing tools users however specify often quite detailed test models, at level of the implementation to be tested. We present here a first attempt to generate tests from more abstract implementation models, which we call architecture models. With only little refinement these models can be used to validate implementations up to interaction accuracy. Our technique is applied to validate that the camera application software for Nokia Series 60 mobile devices conforms to its UML2 architecture model, employing the model-based testing tool Conformiq Qtronic.

1. Introduction

With recent advances in tool technology, model-based testing has become a subject of increasing interest and popularity in industry to save testing cost. Its capability to automatically generate tests from models is envisioned to significantly reduce the need for manual implementation of test suites for the validation of product implementations, e.g., with test scripting languages like TTCN-3 [1, 2].

One of the major challenges in introducing model-based testing to industry is the lack of availability of detailed enough, executable models to derive such tests from. The construction of models at such a level of detail is often as complex as the implementation itself. Many companies are still hesitant to change from conventional techniques to software development with executable modeling specifications since such a change carries a major risk and investments [3].

This paper focuses on the use of model-based design and testing of software systems at a more abstract level, i.e., software architectures. Informal text documents are still widely used to capture functional architecture requirements. Such documents require manual validation of implementations, which is a time consuming and non-trivial task. Deviations from the requirements are however a common reason for problems discovered during software integration.

Architecture models define the behavior of system components at their external interfaces. They do not however model internal computations performed as a result of interactions. Therefore architecture models by themselves are abstract system models which can not be used as-is to validate software implementations. They have to be first refined to be suitable for testing by adding more information about interface data. After such a refinement however generated tests can be executed against real implementations of the model.

We present this approach in the context of the non-trivial camera application used by Nokia Series 60 mobile phones. In our case study the architecture model has been specified after the implementation based on existing interface and design documentation. Although in this case model-based testing has validated the model instead of the implementation, our approach can also be applied in a classical top-down design approach, i.e., to validate implementations against a model.

2. Related research

The phrase "model-based testing" (MBT) has been used to signify many approaches to quality assurance. A good overview and discussion about hopes and reality of MBT can be found in [3]. Our test environment which is presented in the later sections fits the generic architecture shown in this paper. [4] presents an approach which is quite heavily tied to a specific tool, Java as a final implementation language, and software module testing. A lot of work on MBT [5, 14] is based on test models as a basis for test

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

generation, i.e., models of users or the environment that a system is embedded in. Most approaches (with the exception of [14]) have used complete, i.e., data accurate, model specifications as a starting point for test generation.

Our work is founded on the automatic derivation of tests from *system* models [6, 7], i.e., models that describe the desired functionality of the system under test. In this form, MBT as a technology is a derivative of formal conformance testing which has been established as a discipline by Jan Tretmans in [6].

The Lyra modeling method which has been used for the construction of our abstract architecture system model has been first described in [8]. It has been used and studied in modeling SpaceWire network architecture [9] and in the design and specification of a telecommunication protocol [10]. These case studies have focused on modeling aspects and explored formal verification of system model in different design phases.

3. Architecture modeling

Lyra [8] is a method that has been developed for modeling potentially distributed service architectures. The method as such is language independent, but in practice UML2 [11] has been used to create our models. Lyra models cover both, structure and behavior, of a system and are executable. The main motivation for system modeling with Lyra is to ease management and further evolution of system architecture during the lifetime of a product.

Architecture models developed with Lyra describe a system as a set of high-level entities, the interfaces between them, and the externally observable behavior on the interfaces. Behavior is specified at level of interactions and identifies allowed order and sequences of operations on interfaces. Data carried in operation parameters are abstracted and only included in the model if it affects model behavior. Thus, data correctness is not a goal of Lyra architecture models.

A key concept in Lyra is a service component. Service components are used to define the behavior of the system and may communicate with each other via two kinds of interfaces: PSAP (Provided Service Access Point) and USAP (Used Service Access Point). PSAP communication is performed with the user of the service whereas USAP communication is performed with another used service.

The service provided by a service component is described by a hierarchical state machine. The top level PSAP Communication state machine describes the interaction with the user of the service. The actions taken by the service in response to the user requests are encapsulated in composite states, so called Execution Control state machines. Each Execution Control state machine specifies the different steps needed to perform the action requested by the user. If some steps require the use of other services the required interactions again are encapsulated in composite states, so called USAP Communication state machines. Each USAP state machine describes the communication with this other service. Figure 3 shows an example of the hierarchical Lyra state machines.

The main purpose of separating external communication and internal computation to different state machines is to facilitate gradual development of system models. It also allows early validation and verification of model refinement steps.

4. Testing environment

Figure 1 shows an overview of our test environment that is used in order to generate and execute the tests. The test system is based on a model-based testing tool that interacts via adapters with the Implementation Under Test (IUT), i.e., the Series 60 Camera Application software, which is running in a Symbian [12] operating system emulator.

Tests are generated from a refined architecture model based on a set of user defined coverage criteria, i.e., generation directives. The test system communicates with the IUT interfaces via a generic adapter infrastructure which is customized with plug-ins for the specific interfaces used in testing. These plug-ins do not implement any interface behavior but only convert signals generated by the tool to their respective Symbian C++ API calls and vice versa.

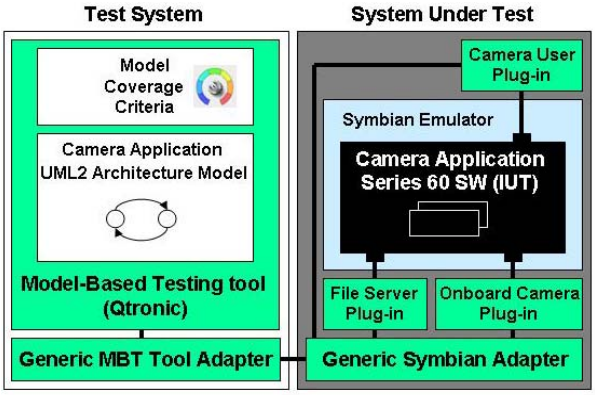


Figure 1. Model-based testing environment

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

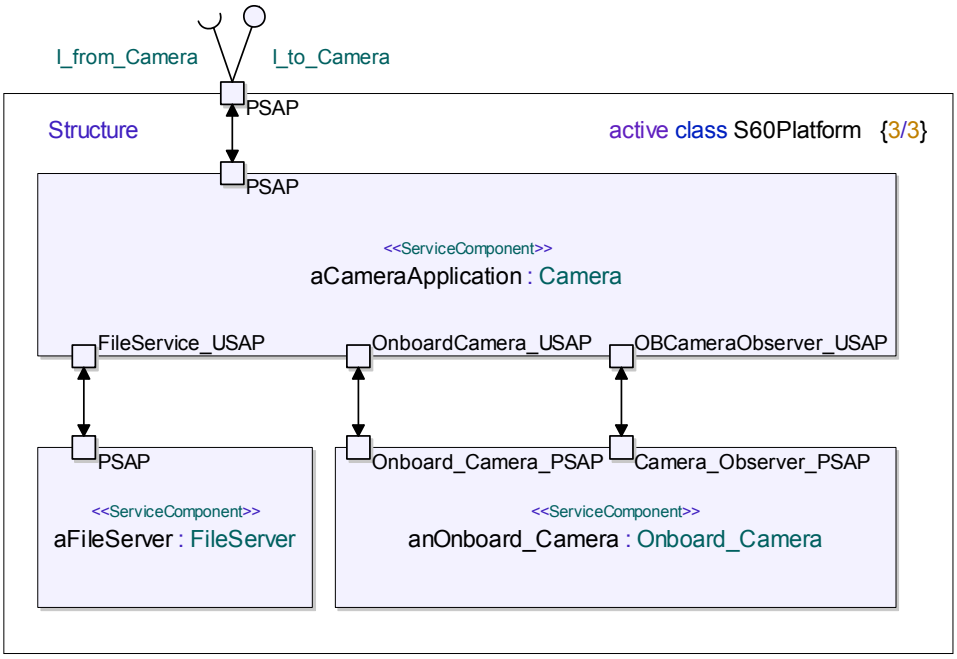


Figure 2: The Camera Application entity within the S60 Platform model

4.1. About the implementation under test

The camera feature and its operation are quite well known to end users of Series 60 camera phones. This feature has been chosen as a case study as it involves four different entities, i.e., user, camera application, camera hardware drivers, and operating system, each of which are implemented and managed by different parties or organizations. In addition, the interfaces between these entities are well known, public, and documented interfaces, i.e., Camera User, Onboard Camera, and File Server APIs. The selected implementation under test is the Camera Application software which processes user input and then either interacts via drivers with the camera hardware or uses operating system file services.

For example, when a user presses a button to capture a picture this request is reported to Camera Application which first requests the picture via the Onboard Camera API and then saves it via the File Server API before displaying it to the user.

The actual implementation of the Camera Application is fairly complex. In our case this entity is decomposed into smaller entities which each have a number of additional internal interfaces. This information is however intentionally not included in the architecture model since it not visible via these above APIs.

4.2. The model-based testing tool

For the generation of tests from refined architecture models we employed the model-based testing tool Conformiq Qtronic [13]. This off-the-shelf tool has not been specifically developed for architecture modeling but can handle virtually any executable UML2 model specification as an input. The primary reason for its selection has been its support for the derivation of tests from system models.

The tool facilitates both, online and offline, model-based testing. In offline tests are generated as external test scripts that then can be subsequently executed independently. Online testing is a more advanced form of model-based testing where tests are executed dynamically "on-the-fly" against the SUT. The fundamental benefit of online over offline testing is that it allows generation of tests from arbitrarily non-deterministic system models.

Qtronic generates tests using a custom engine for symbolic model execution. The symbolic execution engine is used in principle to "invert" the system model so that allowed input behavior of the model becomes allowed output behavior of the model-based test system. The tool can to handle control, timing, as well as data aspects automatically as part of this inversion. Therefore the execution engine is the key feature which it enables us to use system models directly as testing assets.

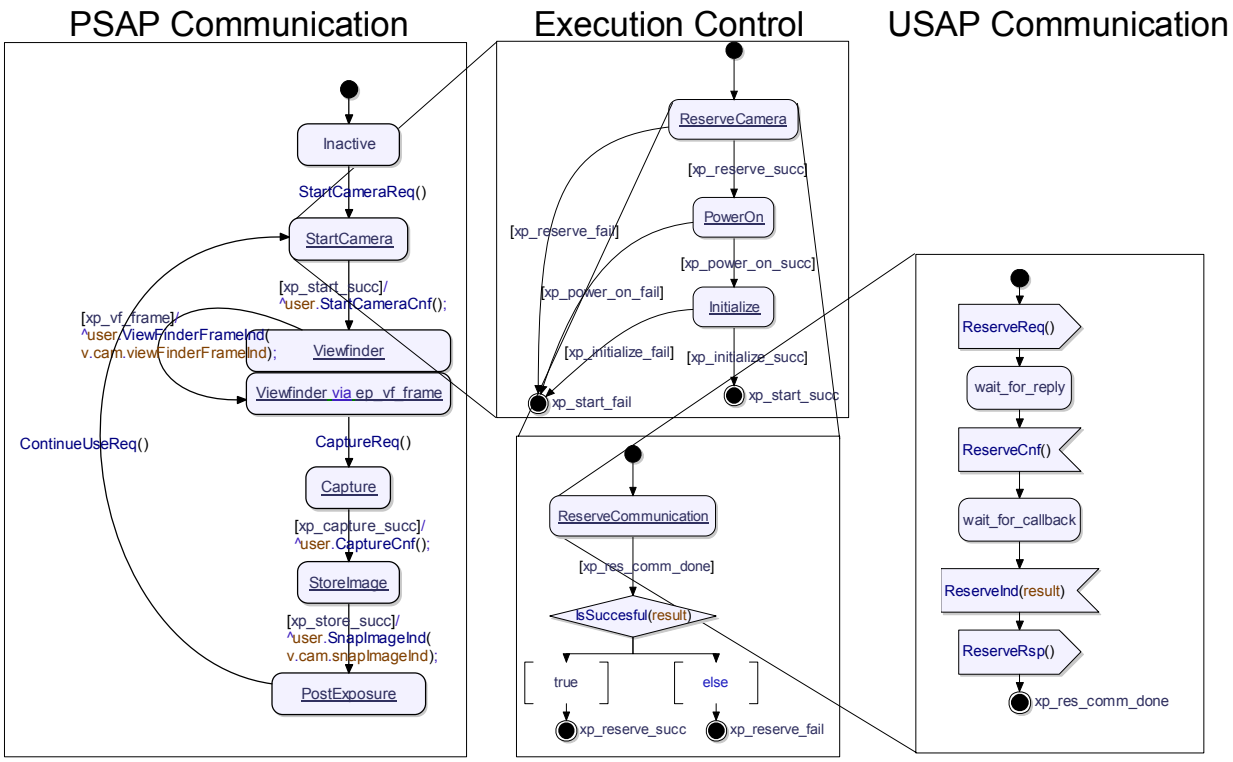


Figure 3: Examples of Lyra state machines specifying Camera Application behavior

In order to guide test selection, this MBT tool allows the user to activate a number of model-based coverage criteria. These include transition coverage (cover all transitions in all state charts), state coverage (cover all states), branch coverage (cover all positive and negative branches in conditional constructs), condition coverage (cover the positive and negative values of individual atomic Boolean conditions), boundary value pattern (factor numerical comparisons into cases around the decision boundaries) and requirements coverage (cover all links to higher-level requirements specifications).

5. The system model

The starting point for model-based testing is the Lyra architecture model of the Camera Application, which has been developed from public interface as well as internal design documentation and its environment as shown in Figure 2. The behavioral part of the Camera Application model shown in Figure 3 specifies still image capture functionality of the Series 60 software but does not specify its implementation, i.e., it focuses on externally observable communication related to the functionality but does not reflect internal Series 60 software design or interfaces.

In our Camera Application model the user interface has been abstracted from actual user interface, i.e., key press sequences or display bitmaps, and is focused on the actions that a user may perform. For example, the camera can be started either via a shortcut or the general application menu. From the Camera Application point of view the action is the same. Thus, we model the starting of the camera as an action `StartCamera`. This approach is commonly used in modeling user interfaces [14].

The file server API was abstracted in a similar manner, i.e., to open, read, write, or close a file. The communication with the camera hardware drivers, i.e., the Onboard Camera entity, follows the actual C++ API as closely as possible. As the modeling of all communication between different components in Lyra is based on asynchronous signaling, C++ method calls are modeled as pairs of signals, one for the call and another for the return.

Figure 3 shows a part of the Camera Application PSAP state machine as well as examples of an Execution Control and USAP state machine. The data sent and expected in the PSAP or USAP communication is essentially completely abstracted, i.e., the architecture model focuses on reflecting the order and sequence of operation invocations - not

correct parameter values. This abstraction of data handling reduces the system model complexity significantly.

6. The refined system model

In validation, our goal is to have the IUT visit states and follow state transitions specified in the architecture model as we desire. But before such a model can be used with a model-based testing tool it has to be modified to be more specific about parameter values used in triggered or expected operation invocations. If the model is used as-is a model-based testing tool could only generate random values which are likely to cause undesired behavior from the IUT.

In principle the refinement of the system model into a model suitable for automated testing is quite straightforward, i.e., by simply manually adding information about valid parameter values to the model wherever it is needed. Due to our choice in MBT technology the addition of such data requires careful thought. Since our tool generates tests by essentially "inverting" the system model the same mindset needs to be followed when refining parameter data.

Whenever the system model specifies an operation invocation the test tool expects to the IUT to make this invocation, and whenever it specifies to expect an operation invocation the test tool makes that invocation. Therefore, parameter values that need to be specified explicitly in the refined model are in and in-out parameter values for expected operation invocations and in-out and out parameter values for operations invoked by the system model. In theory, this model refinement is a still quite complex task since any combination of valid values for operation parameters should be reflected by a correct refinement.

In practice, we can simplify this refinement step by adding only one of all possible parameter value combinations to the system model. Here, specific values need to be specified for parameters to be sent to the IUT and any valid value can be accepted from the IUT, e.g., via the use of subtypes for such a parameter. Suitable parameter values can be determined, e.g., from IUT observation. Note that repetitive behavior in a system model, i.e., loops, need to be unfolded as part of its refinement since correct parameter data values commonly differ between iterations.

Figure 4 shows the refinement of `isSuccessful` function which is invoked to make a decision in the `ReserveCamera` state machine in Figure 3. Here, set of parameter values which should be expected has been restricted to cover only valid error codes. This refinement enforces that any test generated from this Camera Application model will only send valid error codes to the implementation.

In system model:

```
Boolean isSuccessful( in integer errorCode ) {
   switch (errorcode) {
       case ErrNone :      return true;
       default :           return false
  }
}
```

In refined system model:

```
Boolean isSuccessful( in integer errorCode ) {
   switch (errorcode) {
       case ErrNone :      return true;
       case ErrNotFound,
            ErrGeneral,
            ErrNoMemory,
            ErrNotSupported,
            ErrInUse :     return false
  }
}
```

Figure 4: Refinement of `isSuccessful(…)` function for testing purposes.

With this simplification the possibility to validate the correctness of operation parameter values is lost but model refinement becomes significantly easier. After this refinement step the system model has all the information needed for meaningful test generation. But since we have only added an absolute minimum of information this refined model can of course not used as a basis for ensuring parameter data accuracy of the implementation. A more detailed, implementation level system model is needed for that purpose. Note that the model refinement itself is not dependent on Lyra – it is only tied to the concept of an architecture model, i.e., a model which does not model accurately parameter data.

7. Results

Due to a very challenging time frame we had to customize our generic test environment shown in Figure 1 in a number of places, e.g., to exclude the file server API from the system model and therefore from validation of the Camera Application software. These modifications helped us to reduce the complexity of the test environment setup but still allowed us to validate the IUT against its architectural requirements.

Figure 5 shows an example of test sequence which was generated as a result of selected system model coverage criteria. Like many other generated sequences it focuses on the start up of the Camera Application which is the phase of operation with most interface

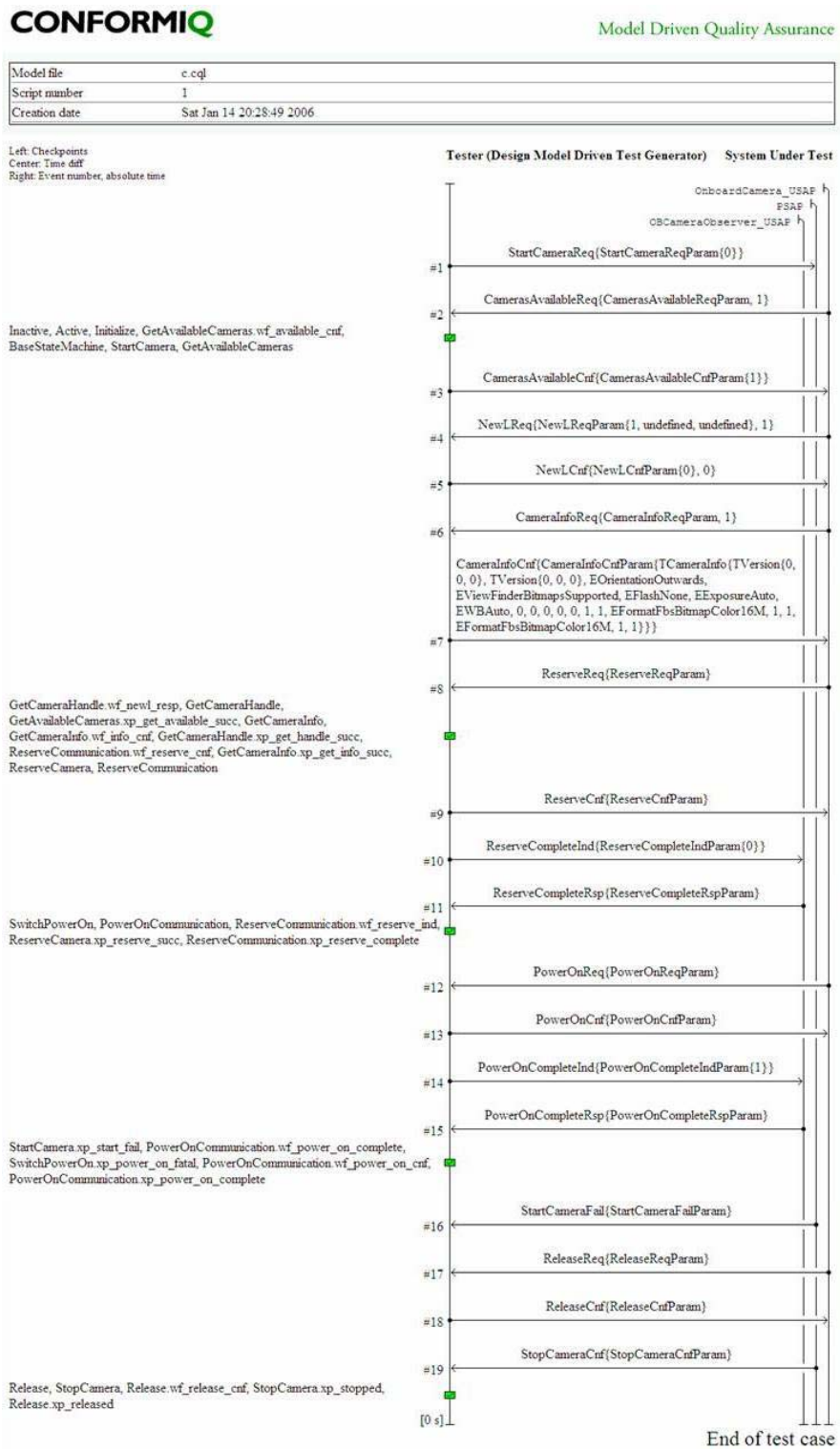


Figure 5: Example of a generated test sequence

accesses. Parameter values shown here as "undefined" correctly specify expected values of in parameters in an operation return. Secondly, parameter lists like `CamerasAvailableReqParam` are empty since they also empty in the interface specification. The trace does not show handling of invalid IUT responses.

The main problem encountered during the architecture model refinement and also later during the execution of generated test sequences has been incomplete or ambiguous design documentation on the use of external services, e.g., missing calls to some operations, as well as handling of erroneous behavior. In some situations these deviations from desired behavior even occurred after including clarifications from the responsible architect in the refinement.

The identification of proper parameter values was mainly performed by observing the IUT in the Symbian emulator and a mock-up user interface. The restriction of some operation parameters to very specific values in UML2 state machines caused some problems, i.e., when a state machine did not branch after the receiving a signal. We solved this issue by introducing a user defined "require" construct which the model based testing tool could then identify as the specification of a specific value.

Finally our experience with coverage criteria has been that the available mechanisms, e.g., to cover each or specific transition(s) or state(s) once, is quite powerful and flexible. However, we feel that more intuitive, simpler techniques for specification of coverage criteria like a MSC [15]-based notation may help users to quickly perform first validations with a new testing environment. In addition, it could help to introduce test engineers who are experienced in conventional suite-based test specification to get familiar with this powerful testing technology more easily.

8. Conclusions

We presented in this paper first results on model-based testing with refined architecture models for software applications. These models focus on the behavior visible at external interfaces of a system. By explicitly excluding the internal structure and interfaces of entities as well as parameter data computation not only the model specification complexity can be decreased significantly. It also leaves designers more freedom in the implementation and evolution of product features, e.g., it allows them to select the programming language of their choice for the implementation of the system. Such models can be developed, e.g., by applying the Lyra modeling method with UML2. Since architecture models are abstract there are not suited as such for use in model-based testing.

We introduced a simple refinement technique to add information about valid parameter values to architecture models. The refined model enables validation of architectural constraints in a real implementation with model-based testing. Tests generated from these models do not however reduce the need for further testing. These tests can, for example, not be used to assess parameter data correctness in interactions via external interfaces. This form of validation however suffices to enforce compliance of an implementation to a given architecture specification.

In the case of the Series 60 Camera Application software results show that the available documentation has not been specific enough to clearly define how external interfaces relate to each other. Our tests have shown that the same documents have been interpreted differently by architects and implementers. Architecture models leave no room for interpretation and offer a means to perform automated model based testing. It has also been shown that model-based testing was able to easily pinpoint the problems with architecture compliance.

Among the tasks for our future work are to further formalize our system model refinement for testing, to develop a proper integrated tool environment to assist test engineers in architecture model refinement, and to continue the study of this technique with other applications. Another area of interest is the investigation of alternative techniques for specification of coverage criteria for model-based testing tools.

Acknowledgements

The authors would like to thank Sami Heinonen, Vesa Luukkala, Tommi Vainikainen, and Kimmo Varpaaniemi for their contributions to the setup of the test environment in a very challenging time frame as well as the reviewers for their helpful comments.

References

[1] C. Willcock et al., *An Introduction to TTCN-3*, Wiley & Sons, 2005.

[2] ETSI European Standard Suite 201 873, *Methods for Testing and Specification (MTS); The Testing and Test Control Notation version 3*, European Telecommunications Standards Institute, Sophia-Antipolis, 2005.

[3] A. Hartma and K. Nagin, "The AGEDIS tools for model based testing", in *Proceedings of the 2004 ACM SIGSOFT international Symposium on Software Testing and Analysis (ISSTA '04)* , Boston, MA, ACM Press 2004, pp. 129-132.

[4] M.J. Rutherford and A.L. Wolf, "A Case for Test-Code Generation in Model-Driven Systems", *Second International Conference on Generative Programming and Component Engineering (GPCE '03)*, LCNS 2830, Springer 2003, pp. 377-396.

[5] P. Gupta, S.J. Cunning, and J.W. Rozenblit, "Synthesis of high-level requirements models for automatic test generation", In *Proceedings of 8th IEEE International Conference and Workshop on the Engineering of Computer Based Systems (ECBS '01)*, Washington, D.C., 2001, pp. 76-82.

[6] J. Tretmans, *A Formal Approach to Conformance Testing*, Ph.D. Thesis, University of Twente, The Netherlands, 1992.

[7] F. Bouquet et al., "Mastering Test Generation from Smart Card Software Formal Models", in *Proceedings of the International Workshop on Construction and Analysis of Safe, Secure and Interoperable Smart devices (CASSIS'04)*, LNCS 3362, Springer 2005, pp. 70-85.

[8] S. Leppänen, *The Lyra Design Method*, Report 40, Institute of Software Systems, Tampere University of Technology, 2005.

[9] J. Honkola, S. Leppänen, and T. Tynjälä, "Modeling the SpaceWire Architecture with Lyra", in *Proceedings of Fifth International Conference on Application of Concurrency to System Design (ACSD)*, St. Malo, France, 2005, pp. 15-24.

[10] S. Leppänen, M. Turunen, and I. Oliver, "Application Driven Methodology for Development of Communicating Systems", in *Proceedings of the ECSI Forum on Specification and Design Languages (FDL)*, Lille, France, 2004.

[11] Object Management Group, *Unified Modeling Language 2.0 Superstructure Specification*, 2005.

[12] www.forum.nokia.com > S60 Platform > Tools/SDKs > S60 Platform SDK

[13] http://www.conformiq.com

[14] A. Kervinen, M. Maunumaa, and M. Katara, "Controlling Testing using Three-Tier Model Architecture", *Electronic Notes in Theoretical Computer Science*, 164 (4), Elsevier 2006, pp. 53-66.

[15] ITU-T Recommendation Z.120 (04/04), *Message Sequence Chart (MSC)*, Geneva, 2004.

Testing time goal-driven requirements with Model Checking Techniques

Gregorio Díaz, Elena Navarro, María-Emilia Cambronero, Valentín Valero and Fernando Cuartero
University of Castilla-La Mancha - Department of Computer Science
Adress: Escuela Politécnica Superior de Albacete. 02071 - SPAIN
Email: [gregorio,enavarro,emicp,valentin,fernando]@dsi.uclm.es

Abstract

In this paper we present the testing of time goal requirements by using Model Checking as an engine for test generation. The goal model that we use to capture the time requirements is an extension of KAOS. This goal model specifies the properties that the system must satisfy and how they are tested by using the model checker engine, UPPAAL. As an illustration of this proposal we take a particular study case based on GSM cell technology.[1]

1 Introduction

In software and hardware development the engineers provide a concrete model of the system behavior. This model is naturally restricted with correctness and performance constraints that have to be checked by the verification software. Another alternative for verifying system is software testing. Software and hardware testing is the process that tests the functionality and correctness of the system through execution or analysis. As noted by Callahan et al in [6] software testing could be used properly in conjunction with model checking to develop high quality software and hardware systems.

Although software and hardware testing can guarantee high quality of software and hardware development, they cannot provide a "correct" system in isolation. On the other hand, the introduction of Formal approaches such as Model Checking, makes this objective feasible. Thus, it becomes important for testing frameworks to incorporate these formal approaches. This has meant a there is growing consensus on the use of formal methods, i.e. development methods based on formalism, which could have significant benefits in testing systems due to the enhanced rigor these methods bring [12]. Furthermore, these formalisms allow us to reason with the constructed models, analyzing and verifying

[1]This work has been supported by the CICYT project "Supported by the spanish government (cofinanced by FEDER founds) with the project "Application of Formal Methods to Web Services", with reference TIN2006-15578-C02-02, and the JCCLM regional project "Application of Formal Methods to the Design and Analysis of Web Services and e-commerce applications", with reference PAC06-0008-6995."

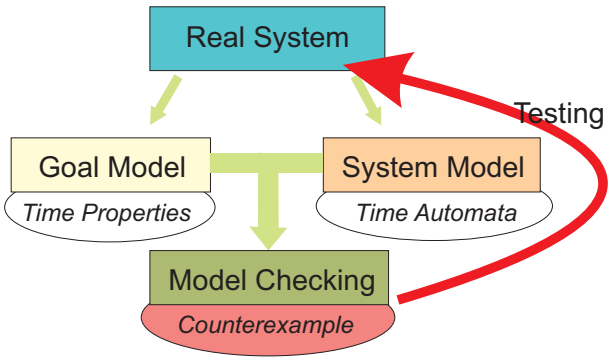


Figure 1. Test Generation Diagram by using Model Checking

some properties of interest of the described systems. One of these formalisms is timed automata [3], which are very frequently used in model checking [1], and there are some well-known tools supporting them, like UPPAAL [11, 8, 14] or KHRONOS [16] either SPIN [13]. The Model Checking performs the verification of models that are derived form the system behaviors. The specification and definition of properties to be verified are an important key. In this work, we use an extension of KAOS [2, 7, 19], an approach for goal-oriented for requirement engineering, in order to formally specify the system requirements. Our approach uses a combination of Model Checking in conjunction with goal-oriented for property gathering. This approach is depicted in figure 1. The goal oriented model and the system model are extracted from the real system and checked by UPPAAL. The checking process may find errors that are model traces that are visualized as counterexamples. These counterexamples are model traces that show paths that the model follows to reach the error. Finally these errors are tested in the real system. Thus this diagram depicts a methodology for test generation by using Model Checking as a tester generation engine.

The paper is structured as follows. In Section 2 we describe the main features of the model checker UPPAAL. The goal model used for the requirement engineering is presented in Section 3. In Section 4 we apply the previous work

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

to a GSM-based communication system, and the UPPAAL tool is used to test and analyze the goal established in the previous one. Finally, the conclusions and the future work are presented in Section 5.

2 Uppaal as a tester engine

UPPAAL is a suite tool that performs symbolic model checking over timed automata. The time properties that it is possible to specify by UPPAAL are based on the temporal logic subset of timed computation tree logic (TCTL). Now, we will see the timed automata, the properties and the tester algorithm in detail.

2.1 Timed Automata

By definition, a timed automaton is a standard finite-state automaton extended with a finite collection of real valued clocks. The clocks are assumed to proceed at the same rate and their values may be compared with natural numbers or reset to 0. UPPAAL extends the notion of timed automata to include integer variables, i.e. integer valued variables that may appear freely in general arithmetic expression used in guards as well as in assignment. Note that for backward reachability analysis, which we will define after, the variable assignment is restricted to any value of the form $ax + b$ where $a, b \in \mathbf{Z}$ and x is the variable being reassigned, whereas for forward reachability analysis there is not such restriction.

The model also allows clocks not only to be reset, but also to be set to any non-negative integer value.

Definition 1 (Atomic Constraints) *Let C be a set of real valued clocks an I a set of integer valued variables. An atomic clock constraint over C is a constraint of the form: $x \sim n$ or $x - y \sim n$, for $x, y \in C$, $\sim \in \{\leq, \geq, =\}$and $n \in \mathbf{N}$. An atomic constraint over I is a constraint of the form: $i \sim n$, for $i \in I$, $\sim \in \{\leq, \geq, =\}$and $n \in \mathbf{Z}$.*

By $C_c(C)$ we will denote the set of all clock constraints over C, and by $C_i(I)$ we will denote the set of all integer constraints over I.

Definition 2 (Guards) *Let C be a set of real valued clocks, and I a set of integer valued variables. A guard g over C and I is a formula generated by the following syntax: $g ::= c|g \wedge g$, where $c \in (C_c(C) \bigcup C_i(I))$.*

$\mathcal{B}(C, I)$ will stand for the set of all guards over C and I.

Definition 3 (Assignments) *Let C be a set of real valued clocks and I a set of integer valued variables. A clock assignment over C is a tuple $\langle v, c\rangle$, where $v \in C$ and $c \in \mathbf{N}$. An integer assignment over I is a tuple $\langle w, d\rangle$ representing the assignment $w = d$, where $w \in I$ and $d \in \mathbf{Z}$.*

We will use $\mathcal{A}(C, I)$ to denote the power-set of all assignments over I and C.

Definition 4 (Timed automata) *A timed automaton A over a finite set of actions $Act, clocksC$ and integer variables I is a tuple $\langle L, l_0, E, V\rangle$, where L is a finite set of nodes (control-nodes), l_0 is the initial node, $E \subseteq L \times \mathcal{B}(C, I) \times Act \times \mathcal{A}(C, I) \times L$ corresponds to the set of edges, and $V : L \rightarrow \mathcal{B}(C, I)$ assigns invariants to locations. For a brief notation, we will denote $l \xrightarrow{g,a,r} l'$ by the edge $\langle l, g, a, r, l'\rangle \in E$.*

Definition 5 (Synchronization Function) *Let $\mathcal{T} \subseteq Act \times Act$ be a function such that:*

$$\langle a_i!, a_i?\rangle \in \mathcal{T} \Rightarrow \langle a_i?, a_i!\rangle \in \mathcal{T} \text{ for all } a_i$$

Definition 6 (Parallel Composition) *Let A_1, A_2 be two timed automata. The parallel composition $(A_1|A_2)$ is a timed automaton $\langle L, l_0, E\rangle$, where $(l_1|l_2) \in L$ whenever $l_1 \in L_1$ and $l_2 \in L_2$, $l_0 = (l_{1,0}|l_{2,0})$, and the edges E are defined as follows:*

- $(l_1|l_2) \xrightarrow{g,\tau,r} (l_1'|l_2')$ *if* $(l_1 \xrightarrow{g_1,c!,r_1} l_1') \wedge (l_2 \xrightarrow{g_2,c?,r_2} l_2') \wedge (g = g_1 \wedge g_2) \wedge (\langle c!, c?\rangle \in \mathcal{T}) \wedge (r = r_1 \cup r_2)$
- $(l_1|l_2) \xrightarrow{g,a,r} (l_1'|l_2)$ *if* $(l_1 \xrightarrow{g,a,r} l_1')$
- $(l_1|l_2) \xrightarrow{g,a,r} (l_1|l_2')$ *if* $(l_2 \xrightarrow{g,a,r} l_2')$

A **state** of a timed automaton A is a pair $\langle l, u\rangle | u \in V(l)$, where l is a node of A and u is an assignment, mapping each clock in C to a value in $\mathbf{R}_+$, and each integer variable in I to a value in $\mathbf{Z}$, and $u \in V(l)$ means that u satisfies the invariant of the node l. We will use $g(u)$ to denote that the assignment u satisfies the guard g. The initial state of A is $\langle l_0, u_0\rangle$, where u_0 is the assignment mapping all variables and clocks to 0.

The evolution of a timed automaton, from a state to another state can proceed by two types of transitions:

- Delay transition: $\langle l, u\rangle \xrightarrow{\in(d)} \langle l, u'\rangle$.
- Action transition: $\langle l, u\rangle \xrightarrow{g,a,r} \langle l', u'\rangle$.

Definition 7 (Delay transitions) *Let $\langle l, u\rangle$ and $\langle l', u'\rangle$ be two states of a timed automaton A, and let d be a positive real. Then*

$$\langle l, u\rangle \xrightarrow{\in(d)} \langle l', u'\rangle \text{ iff } u' \in V(l') \wedge \begin{cases} l' = l & \\ u'(x) = u(x) + d & \text{if } x \in C \\ u'(x) = u(x) & \text{if } x \in I \end{cases}$$

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Definition 8 (Action transition) *Let $\langle l, u\rangle$ and $\langle l', u'\rangle$ be two states of a timed automata A. Then*

$$\langle l, u\rangle \xrightarrow{g,a,r} \langle l', u'\rangle \text{ iff } u' \in$$
$$V(l') \wedge g(u) \wedge \left(u'(x) = \begin{cases} c_0 & \text{if } x \in C \wedge \langle x, c_0\rangle \in r \\ c_1 & \text{if } x \in I \wedge \langle x, c_1\rangle \in r \\ u(x) & otherwise \end{cases} \right)$$

2.2 Specifying time properties

The language that UPPAAL uses to perform the verification is a subset of timed computation tree logic (TCTL) [18, 17], where atomic expressions are location names, variables and clocks from the modeled system. The properties are defined using local properties that are either true or false depending on a specific configuration.

Definition 9 (Local Property) *Given an UPPAAL model $\langle \vec{A}, Vars, Clocks, Chan, Type\rangle$. A formula φ is a local property iff it is formed according to the following syntactical rules:*

$\varphi ::=$	*deadlock*	
$\mid$	$A.l$	*for* $A \in \vec{A}$ *and* $l \in L_A$
$\mid$	$x \bowtie c$	*for* $x \in Clocks$, $\bowtie \in \{<, <=, ==, >=>\}, c \in \mathbb{Z}$
$\mid$	$x - y \bowtie c$	*for* $x, y \in Clocks$, $\bowtie \in \{<, <=, ==, >=>\}, c \in \mathbb{Z}$
$\mid$	$a \bowtie b$	*for* $a, b \in Vars \bigcup \mathbb{Z}$, $\bowtie \in \{<, <=, !=, ==, >=>\}$
$\mid$	(φ_1)	*for* φ_1 *a local property*
$\mid$	*not* φ_1	*for* φ_1 *a local property*
$\mid$	φ_1 *or* φ_2	*for* φ_1, φ_2 *logical properties (logical OR)*
$\mid$	φ_1 *and* φ_2	*for* φ_1, φ_2 *logical properties (logical AND)*
$\mid$	φ_1 *imply* φ_2	*for* φ_1, φ_2 *logical properties (logical implication)*

In Definition 9 we have expressed the syntaxis of the temporal logic that UPPAAL uses. Now, let us see the definition of the five different property classes that UPPAAL may test.

Definition 10 (Temporal Properties) *let $M = \langle \vec{A}, Vars, Clocks, Chan, Type\rangle$ be an UPPAAL model and let φ and ψ be local properties. The correctness of temporal properties is defined for the classes $A[\ ]$, $A <>$ and $-->$ as follows:*

$M \vDash A[\ ]\,\varphi$	*iff*	$\forall \{(\vec{l}, e, v)\}^K \in \tau(M).$ $\forall k \leq K.\ (\vec{l}, e, v)^k \vDash_{loc} \varphi$
$M \vDash A <> \varphi$	*iff*	$\forall \{(\vec{l}, e, v)\}^K \in \tau(M).$ $\exists k \leq K.\ (\vec{l}, e, v)^k \vDash_{loc} \varphi$
$M \vDash \varphi --> \psi$	*iff*	$\forall \{(\vec{l}, e, v)\}^K \in \tau(M).\ \forall k \leq K$ $(\vec{l}, e, v)^k \vDash_{loc} \varphi \Rightarrow \exists k' \geq k.$ $(\vec{l}, e, v)^{k'} \vDash_{loc} \psi$

The two temporal property classes dual to $A[\ ]$ and $A <>$ are defined as follows:

$M \vDash E <> \varphi$	*iff*	$\neg(M \vDash_{loc} A[\ ]\ not(\varphi))$
$M \vDash E[\ ]\,\varphi$	*iff*	$\neg(M \vDash_{loc} A <>\ not(\varphi))$

2.3 The tester algorithm

The tester algorithm that UPPAAL uses is based on the symbolic model checking [4, 15] that uses constraint solving. The algorithm checks if a state in a timed automata is reachable from the initial state or not.

When searching the state space we need two buffers that we can call "wait" and "passed" respectively. The wait buffer holds the states not yet explored and the passed buffer holds the states explored so far.

Algorithm 1 ***Forward Reachability Analysis***

If we do forward reachability analysis we initially store $\langle l_0, U_0\rangle$ in the wait buffer. We then repeat the following:

1. *Pick a state $\langle l_i, U_i\rangle$ from the wait buffer.*
2. *Check if $l_i = l_f \wedge U_i \subseteq U_f$. If that is the case, return the answer yes.*
3. *If $l_i = l_j \wedge U_i \subseteq U_j$, for some $\langle l_i, U_i\rangle$ in the passed buffer, drop $\langle l_i, U_i\rangle$ and go to step 1. Otherwise save $\langle l_i, U_i\rangle$ in the passed buffer. If $U_j \subset U_i$ we can replace the state $\langle l_j, U_j\rangle$ with $\langle l_i, U_i\rangle$. (To save space)*
4. *Find all l_k that are reachable from l_i in one step regardless of the assignments, taking only actions into account. Let g_k be the set of guards on the performed transition an a_k the set of resets*
5. *Now set $U_k = reset(sp(U_i) \cap g_k, a_k)$. If $U_k \neq \emptyset$, store $\langle l_k, U_k\rangle$ in the wait buffer.*
6. *If the wait buffer is not empty go to step 1, otherwise return the answer no.*

3 The Goal Model for time requirements

The requirements, properties and characteristic of the system must be gathered in order to be tested. However, they must be expressed in a formalized manner. There are several languages, graphical diagrams, etc. to perform it,

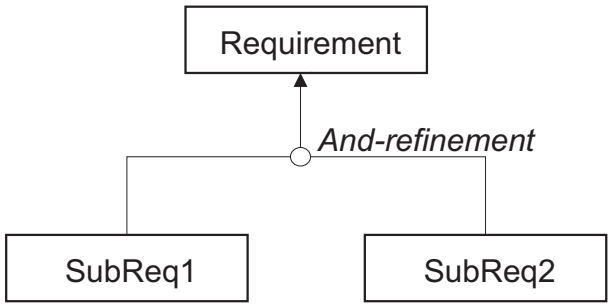


Figure 2. And-refinement goal model

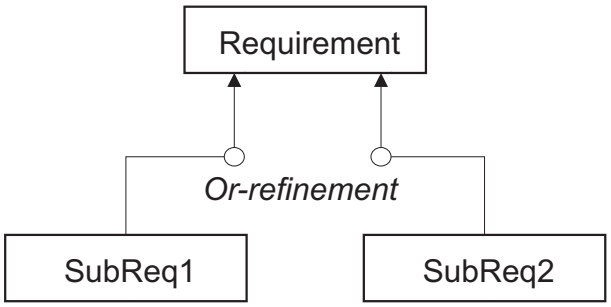


Figure 3. Or-refinement goal model

but we apply those in which the time requirements are well captured. In this sense, goal oriented requirements engineering emerges as a natural choice.

The key activity in goal-oriented requirements engineering is the construction of the goal model. Goals are objectives the system under construction must achieve. Goal formulations thus refer to intended properties to be ensured. They are formulated at different levels of abstraction from high-level, strategic concerns to low-level technical concerns.

Goal models also allow analysts to capture and explore alternative refinements for a given goal. The resulting structure of the goal model is an AND-OR graph. The specific goal-oriented framework considered here is an extension of KAOS methodology [2, 5, 7, 19] which has a two-level language: (1) an outer semi-formal layer for capturing, structuring and presenting requirements engineering concepts; (2) an inner formal assertion layer for their precise definition and for reasoning about them.

Two key elements are used as building elements for the definition of a goal model: *goals* and *requirements*. A goal prescribes intended behaviors of the system. It may refer to services to be provided (functional goals) or to the quality of service (non-functional goals). A requirement is a leaf goal that requieres the cooperation between different parties, which are called *agents*. Agents are active components that play a role in achieving goal satisfaction.

To Build Goal Models, goals are organized in an AND/OR refinement - abstraction hierarchy where higher-level goals are, in general, strategic, coarse-grained and involve multiple agents whereas lower-level goals are, in general, technical, fine-grained and involve fewer agents. In such structures, AND-refinement links relate a goal to a set of subgoals (called refinement) possibly conjoined with domain properties; this means that satisfying all subgoals in the refinement is a sufficient condition in the domain for satisfying the goal, as seen in Figure 2. OR-refinement links may relate a goal to a set of alternative refinements, as seen in Figure 3.

Requirements must be checked by the model checker. During the check process, if the system model does not satisfy a requirement then this requirement must be tested over the real system using the counterexample (see figure 1). Requirements are formalized in a real-time temporal logic that we have shown above. Keywords such as *Achieve* (reachability), *Avoid* (not safety), *Maintain* (safety), *possibly always*, *inevitably* and *unbounded response*, are used to name goals according to the temporal behavior pattern they prescribe. They are depicted in the goal model as follows:

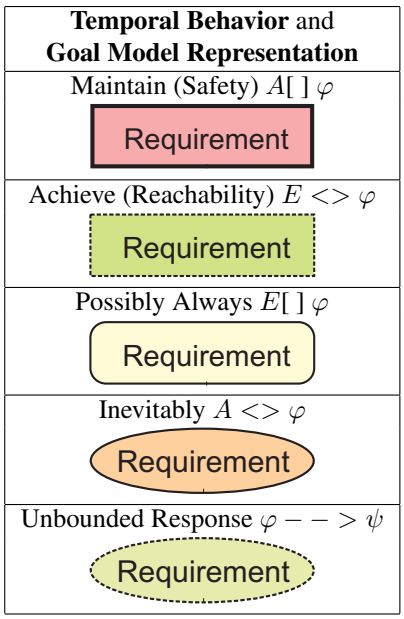


Figure 4 shows a goal model fragment of an aircraft control system. The goal Maintain (safety) *[SafeDoors]* is refined by two *OR-refinement* links that inherit the safety behavior. The first leaf goal *Maintain[DoorsLockedWhileMoving]*, may be annotated with the following temporal logic assertion stating that in every future state the plane doors shall be locked when the plane is moving. The *[SafeDoors]* formulation will be:

$$A[\,]\,(Plane.Moving \wedge Doors.Locked).$$

The second leaf goal: *Maintain Inevitably [DoorUnlockedWhenEmergency]*, may be annotated with the following temporal logic assertion stating that in every future state the plane doors shall be unlocked when emergency occurs:

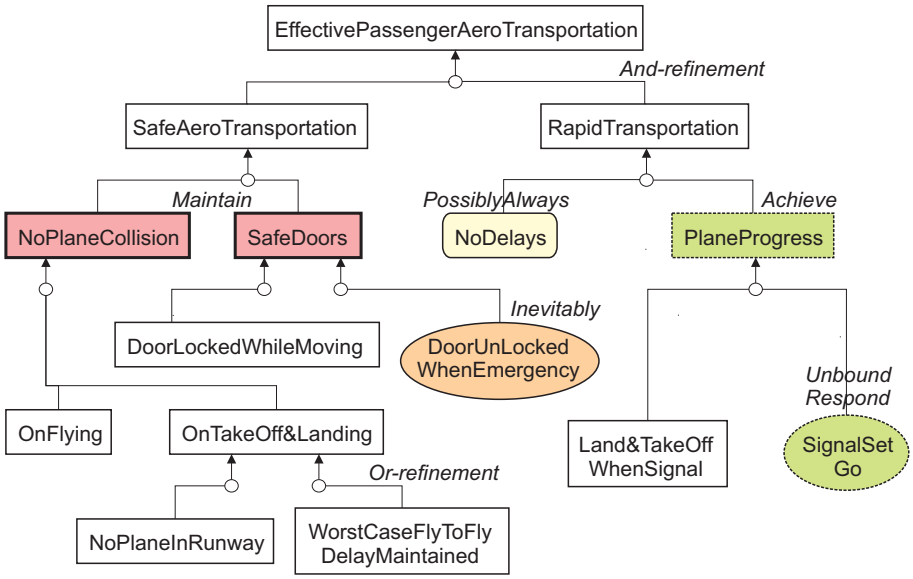


Figure 4. Portion of a goal graph for an aircraft control system

$$A <> Plane.Emegency \wedge Doors.Unlocked.$$

The alternative refinement for the safety subgoal *[NoPlaneCollisionOnTakeOff&Landing]* which leads to totally different designs when refined: block-based design for *NoPlainInSameRunway* and speed control design for *WorstCaseFlyToFlyDelayMaintained*, whose definitions are, respectively:

$$A[]\ \neg(Runway.Ocduppied) \text{ and}$$

$$A[]\ Runway.FlyToFlyClock < FlyToFlyDelay$$

The leaf goal *[NoDelays]* specifies the intended behavior of "Possibly Always, the aircraft control does not suffer delays" which is defined by the formula:

$$E[]\ \neg(AircraftControlSystem.Delay)$$

The temporal behavior for plane progression is specified by the Achieve (reachability) subgoal *[PlaneProgress]* which is refined by two subgoals that inherit the achieve behavior: *[Land&TakeOffWhenSignal]* and *[SignalSetGo]* defined by

$$E <> (Plane.LandorPlane.TakeOff) \wedge Signal.Go$$
and
$$E <> Plane.Ready - - > Signal.Go$$

Goals refer to objects or entities which may be incrementally designed from goal specifications to produce a structural model of the system that is specified as an automata. Objects have states defined by the values of their attributes and associations to other objects. In the above formalization of the goal *DoorsLockedWhileMoving*, Moving and *doorsState* are attributes of the Train and doors entities that will be declared in the system design.

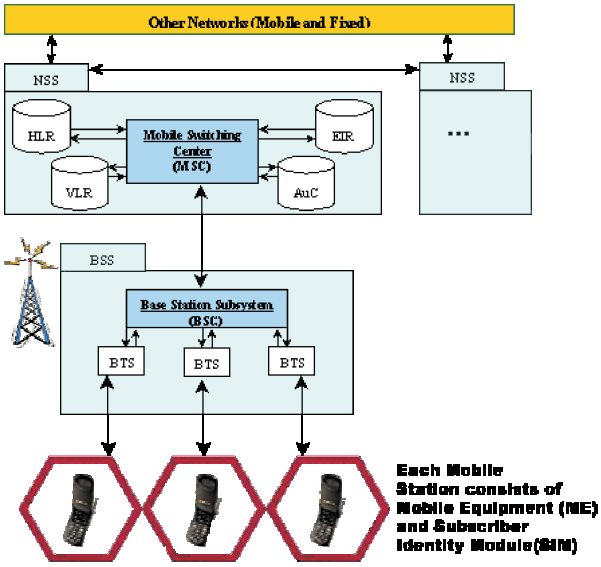


Figure 5. The GSM architecture

4 Case Study

Our example is based on the Global System for Mobile communications (GSM) and centred on two components of the GSM cell architecture, Mobile Stations and Base Station Subsystem. A GSM is composed of several funtional entities, whose functions and interfaces are specified in Figure 5.

Mobile Station, MS for short, consists of the mobile equipment (the terminal) and a smart card called the Subscriber Identity Module (SIM). The SIM provides personal mobility, so that the user can have access to subscribed services irrespective of a specific terminal. By inserting the

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

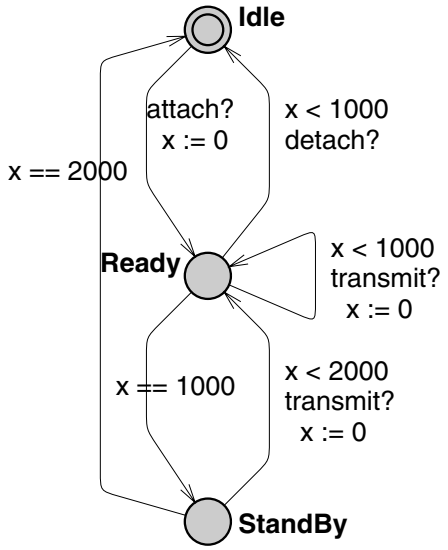


Figure 6. The MS automaton

SIM card into another GSM terminal, the user is able to receive calls at that terminal, make calls from that terminal, and receive other subscribed services.

The mobile equipment is uniquely identified by the International Mobile Equipment Identity (IMEI). The SIM card contains the International Mobile Subscriber Identity (IMSI) used to identify the subscriber to the system, a secret key for authentication, and other information. The IMEI and the IMSI are independent, thereby allowing personal mobility. The SIM card may be protected against unauthorized use by a password or personal identity number.

The Base Station Subsystem, BSS for short, is composed of two parts, the Base Transceiver Station (BTS) and the Base Station Controler (BSC). These communicate across the standardized Abis interface, allowing (as in the rest of the system) the operation between components made by different suppliers.

The Base Transceiver Station houses the radio tranceivers that define a cell and handles the radio-link protocols with the Mobile Station. In a large urban area, there will potentially be a large number of BTSs deployed; thus the requirements for a BTS are ruggedness, reliability, portability and minimum cost.

The Base Station Controler manages the radio resources for one or more BTSs. This includes radio-channel setup, frequency hopping and handovers, as described below. The BSC is the connection between the mobile station and the Mobile service Switching Center (MSC).

4.1 Timed Automata Specification

This automata specification deals with the relationship between MS and BTS components. It captures the behavior of a data transmission from a BTS component to a MS component. This transmission process often has time restrictions that determine the time interval within which the communication is available. Figures 6 and 7 depict the automata that captures the MS and BTS behaviors, respectively. Figure 6 captures the MS behavior that consists of three states interconnected by transitions that define the communication protocol with a BTS partner. Figure 7 is the specification automaton of a BTS component that consists of two states interconnected also by several transitions.

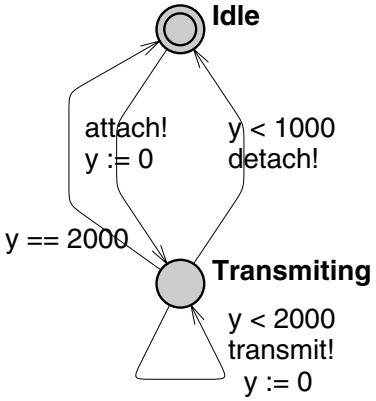


Figure 7. The BTS automaton

Basically, the example shows a MS component that is waiting to receive an attach message from a BTS component. Once the MS has received this message, it is ready to receive a data transmission from this BTS component. If no data from the BTS is received within 1000 milliseconds then the MS changes its state to StandBy. Furthermore, if no data is incoming in another interval of 1000 milliseconds then the MS changes to idle and the process starts again. The BTS automaton may transmit by using the transmit message or finishes the transmission by using the detach message and then the BTS and MS automata changes to idle at the same time.

An important tool for capturing the temporal behavior is the clocks, and in this example, we have used two clocks, x and y. These clocks are used in guards, invariants and assignments. Guards are placed on transitions and determines when a transition is enabled. For instance, we can observe in our example the guard in the transition that connects the "StandBy" and "Idle" states of the MS automaton in figure 6. This guard enables the tansition when the x clock reaches 2000 time units. Invariants are placed on states and specify the time that it is possible to stay in this state. Assignments are placed on transitions and modify either a variable or a clock when the transition is made. We also can observe an assignment in the transition between the "Idle" and "Transmitting" states in the BSS automaton in figure 7. This assignment resets the clock x when the transition is carried out.

The channels are other components of our automata and

also a key component to specify communications. For instance, in this case we have used three channels attach: transmit and detach. The symbols "!" and "?", which are placed after channels, determine that the channel is being used to synchronize two automata. The synchronization party that uses the "!" symbol is the one that initialize the communication and the one marked with "?" symbol receives the communication. Note that the communication must be established by using the same channel and no other. Thus the communication is performed when a pair of "!" and "?" symbols appear over the same channel.

4.2 Testing time requirements

Figure 8 depicts the goal model for the time requirements that our example must satisfy. The root goal [CorrectCellSystem4GSM] is divided in two subgoals by using an AND-refinement. The first subgoal is [PropperBTSBehavior] and the second one is [PropperMSBehavior].

The *[PropperBTSBehavior]* subgoal has the temporal character of a maintain goal, which implies that it must always be satisfied. This subgoal consists of an OR-refinement with an other two leaf goals *[BTSTransmitsWhithinTheInterval]* and *[BTSNoTransmitsOutTheInterval]*, which inherit the maintain character from *[PropperBTBehavior]*. Note that the OR-refinement stipulate that it is enough if only one of these leaf goals is satisfied in order to satisfy the super goal *[PropperBTSBehavior]*.

The *[BTSTransmitsWithinTheInterval]* leaf goal specifies that the temporal property of the BTS component always transmits data within the interval of 2000 milliseconds and is specified as follows:

$$\forall\Box(BTS.Transmit \Rightarrow BTS.y <= 2000) \qquad (1)$$

The *[BTSIdleOutTheInterval]* leaf goal specifies that the temporal property of the BTS component is idle (or does not transmits data) out the interval of 2000 milliseconds and is specified as follows:

$$\forall\Box(BTS.y > 2000 \Rightarrow BTS.Idle) \qquad (2)$$

The *[PropperMSBehavior* subgoal has the temporal character of an achieve goal and consists of an AND-refinement in two subgoals *[PropperMSReception]* and *[PropperMSStops]* that inherit the achieve character. The *[PropperMSReception]* subgoal is divided by an AND-refinement in two leaf goals with an unbounded response character that are labeled as *[MSWillBeReadyOnTime]* and *[MSSTandByForTheInterval]*. The *[PropperMSStops]* consists of an AND-refinement with two unbounded response characterized leaf goals named *[MSStopsWhenDetach]* and *[MSStopsWhenNoData]*.

The *[MSWillBeReadyOnTime]* leaf goal states that the property of the MS will be ready on time to receive data from a BTS component and is specified as follows:

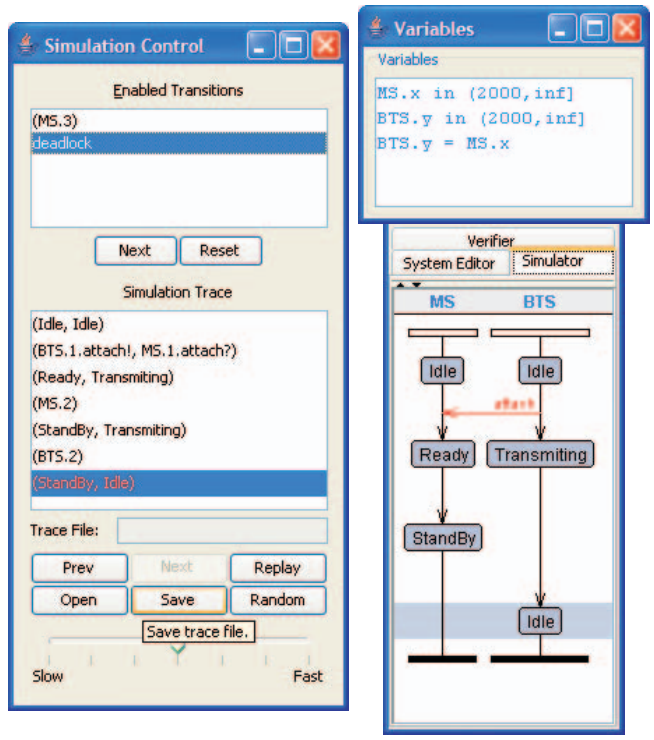


Figure 9. The counterexample trace generated by UPPAAL

$$MS.Idle -->$$
$$((MS.Ready \wedge MS.x <= 1000) \vee MS.Idle) \qquad (3)$$

The *[MSStandByForTheInterval]* leaf goal states that the property of the MS will stay in the Stand By state during 1000 milliseconds if no data is transmitted and is specified as follows:

$$MS.Ready -->$$
$$((MS.StandBy \wedge MS.x >= 1000 \wedge MS.x <= 2000)$$
$$\vee MS.Idle \vee MS.Ready) \qquad (4)$$

The *[MSStopsWhenDetach]* leaf goal states that the property of the MS will stop when receives the detach message and is specified as follows:

$$MS.Ready --> ((MS.Idle \wedge MS.x > 1000)$$
$$\vee MS.Ready \vee MS.StandBy) \qquad (5)$$

The *[MSStopsWhenNoData]* leaf goal defines the property of the MS will be ready on time to receive data from a BTS component and is specified as follows:

$$MS.StandBy -->$$
$$((MS.Idle \wedge MS.x >= 2000) \vee MS.StandBy) \qquad (6)$$

After the specification of these six leaf goals, they must be tested in the timed automata that we have seen in Figures

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

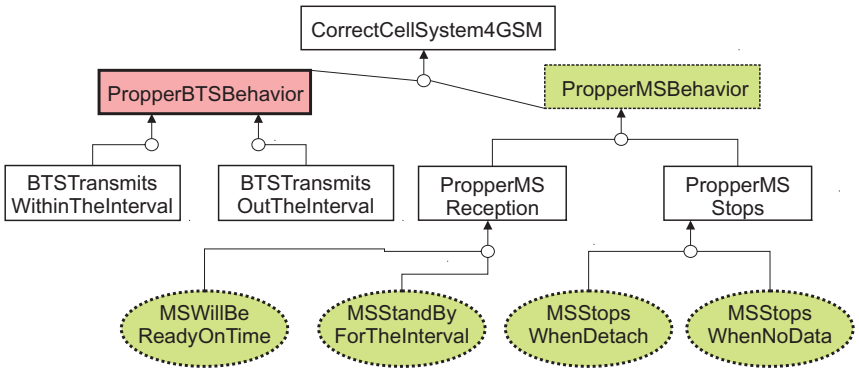


Figure 8. The goal model of the GSM example

6 and 7. To test the automata we use the model checker UPPAAL and the output is *"The five first properties 1, 2, 3, 4 and 5 that represent some leaf goals of the goal model depicted in figure 8 are satisfied by the automata. On the other hand, the last one, 6, is not satisfied"*. By analyzing this last property the counterexample that UPPAAL provides shows that the error is found when the Mobile Station component waits in the state *Stand By* for more than 2000 milliseconds and then a deadlock occurs in the model. As an illustration of this analysis Figure 9 shows the trace of this counterexample generated by UPPAAL.

This conclusion shows us which there are errors in our system model. Thus, this last property 6 is a good test must be checked in the real GSM scenario.

5 Conclusions and Future Work

The use of model checking in the test generation provides the testing of software and hardware with a powerful tool that minimizes its cost. One key point, in this sense, is the automatic generation of counterexamples helpful for testing real systems. In addition, it allows us to minimize the temporal cost of an extensive testing so that the time to make a complete test over the real system is greater than over the system model.

Furthermore, the use of goal-oriented requirement engineering during the specification of properties provides us with an easy way of establishing the requirements that the system must satisfy. It allows us to easily establish that they are the properties to be checked by the verification software.

Thus intertwining these two techniques, model checking and goal-oriented requirement engineering, enhances the software and hardware testing by introducing to it mathematic rigor and a proper requirements description, respectively.

As future work, we are working on the application of these techniques to other fields like Web Services in the Internet application world as mentioned in [10] and [9].

References

[1] *Model Checking*. MIT Press, 1999.

[2] A. van Lamsweerde A. Dardenne and Stephen Fickas. Goal-directed requirements acquisition. page 350.

[3] R. Alur and D. Dill. Automata for modeling real–time systems. In *In Proceedings of the 17th International Colloquium on Automata, Languages and Programming*, volume 443. Springer–Verlag, 1990.

[4] Johan Bengtsson and Wang Yi. Timed automata: Semantics, algorithms and tools. In W. Reisig and G. Rozenberg, editors, *In Lecture Notes on Concurrency and Petri Nets*. Springer-Verlag, 2004.

[5] A. Rifaut J.F. Molderez A. van Lamsweerde C. Ponsard, P. Massonet and H. Tran Van. Early verification and validation of mission critical systems. In *Ninth International Workshop on Formal Methods for Industrial Critical Systems*, 2004.

[6] J. Callahan, F. Schneider, and S. Easterbrook. Automated software testing using modelchecking. In *Proceedings 1996 SPIN Workshop*, August 1996. Also WVU Technical Report #NASA-IVV-96-022.

[7] R. Darimont and A. van Lamsweerde. Formal refinement patterns for goaldriven requirements elaboration.

[8] J. Pardo F. Cuartero G. Diaz, K.G. Larsen and V. Valero. An approach to handle real time and probabilistic behaviors in e-commerce: Validating the set protocol. In *In proceedings of the 2005 ACM Symposium on Applied Computing*.

[9] M. E. Cambronero V. Valero G. Diaz, J. J. Pardo and F. Cuartero. 1automatic translation of ws-cdl choreographies to timed automata. In *In proceedings of WS-FM, 2005*.

[10] M. E. Cambronero V. Valero G. Diaz, J. J. Pardo and F. Cuartero. Verification of web services with timed autoamata. In *In proceedings of First International Workshop on Automated Specification and Verification of Web Sites*.

[11] V. Valero G. Diaz, F. Cuartero and F. Pelayo. Automatic verification of the tls handshake protocol. In *In proceedings of the 2004 ACM Symposium on Applied Computing*.

[12] Constance Heitmeyer and Dino Mandrioli. *Formal Methods for Real-Time Computing*. John Wiley & Sons, 1996.

[13] Gerard J. Holzmann. *The SPIN Model Checker : Primer and Reference Manual*. Addison-Wesley Professional, 2003.

[14] K. Larsen, P. Pettersson, and Wang Yi. Uppaal in a nutshell. *Journal on Software Tools for Technology Transfer*, 1, 1997.

[15] Kim G. Larsen, Paul Pettersson, and Wang Yi. Compositional and Symbolic Model-Checking of Real-Time Systems. In *Proc. of the 16*th *IEEE Real-Time Systems Symposium*, pages 76–87. IEEE Computer Society Press, Dec 1995.

[16] O. Maler A. Olivero S. Tripakis M. Bozga, C. Daws and S. Yovine. Kronos: A model-checking tool for real-time systems. In *In Proc. 1998 Computer-Aided Verification, CAV'98*, volume 427. LNCS.

[17] Costas Courcoubetis Rajeev Alur and David L. Dill. Model-checking in dense real-time. In *Journal of Information and Computation*, 1993.

[18] J. Sifakis T.A. Henzinger, X. Nicollin and S. Yovine. Symbolic model checking for real-time systems. In *In Proceedings of the IEEE Conference on Logics in Computer Science (LICS)*, 1992.

[19] A. van Lamsweerde. Requirements engineering in the year 00: a research perspective. In *International Conference on Software Engineering*, page 519, 2000.

Automatic Verification and Performance Analysis of Time-Constrained SysML Activity Diagrams

Yosr Jarraya† Andrei Soeanu† Mourad Debbabi†
†Computer Security Laboratory, Concordia Institute for Information Systems Engineering,
Concordia University, Montreal, Canada.

Fawzi Hassaïne‡
‡Future Forces Synthetic Environments Section,
Defence Research and Development Canada, Ottawa, Ontario, Canada.

Abstract

We present in this paper a new approach for the automatic verification and performance analysis of SysML activity diagrams. Since timeliness is important in the design and analysis of real-time systems, we annotate activity diagrams with time constraints. In order to apply the model checking technique, we use discrete-time Markov chains (DTMC) as a semantic interpretation of such SysML models wherein communication is restricted to synchronization. Thus, we describe a mapping procedure of SysML activity diagrams to their corresponding DTMC and use PRISM model checker for the assessment and evaluation of performance characteristics. Finally, we apply our methodology on a real-life case study meant to assess a systems engineering behavioral model of a photo-camera device.

1 Introduction and Motivations

Systems engineering (SE) has tremendously evolved over the last decades in order to handle increasingly complex systems. Many of the modern systems represent an aggregation of other sub-systems and components each of which might have different expected though not exactly determined characteristics and features. To enable a successful model-driven approach to SE design and development process, the standardization body of OMG has officially released the System Modeling Language (SysML) specification [10]. Based on UML 2.0 but extending and adapting it to fit SE modeling requirements, SysML is expected to have an auspicious future as a standard SE modeling language. One of the main objectives of SE is the realization of successful systems [13]. Thus, verification and validation (V&V) represents an important process that aims at the assessment of the quality of the engineered system and its compliance with the requirements. Particularly, V&V of design models enables the reduction of the risk involved in the engineering of complex systems by allowing the early discovery of faults in the system before actually implementing it. As today's systems are increasingly complex and may tend to exhibit features such as concurrency and probabilistic behavior, traditional V&V techniques (e.g. simulation) are hardly sufficient to enable a complete and rigorous assessment. Consequently, there is an apparent need for handling and verifying aspects such as timeliness and probabilistic behavior. Though some of these issues have been heretofore addressed in the context of software engineering [2, 3, 15, 16], to the best of our knowledge, no such initiatives are targeting SE design models expressed in SysML.

SysML activity diagram is particularly relevant to SE due to its suitability for functional flow modeling commonly used by systems engineers [4]. In fact, it is similar to the Extended Functional Flow Block Diagrams (EFFBDs) [5]. SysML specification has redefined and widely extended activity diagrams using UML profiling mechanism. The main extensions concern the support of continuous and probabilistic systems modeling. SysML activity is used to highlight the inputs, the outputs, sequences, and conditions for coordinating behaviors in the system [10]. Particularly, these behaviors may require time duration to execute and terminate. Thus, we need to specify such constraints in order to perform quantitative analysis and verify time-related properties, especially for real-time systems (e.g. industrial manufacturing control, robotics, and various embedded systems) many of which need to be engineered under strict functional performance requirements.

However, annotation of time constraints on top of SysML activity diagrams is not clearly defined in the standard [10]. The only existing extension with performance and time aspects can be found in the UML Profile for

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Schedulability, Performance, and Time (SPT) [9] adopted for UML 1.4. Nevertheless, SysML did not import the SPT profile and in order to adopt it one should upgrade it accordingly. In fact, SysML specification [10] gives a mere recommendation to use the simple time model defined in [11] to annotate activity diagrams. This model is a UML 2.x sub-package related to the *CommonBehavior* package and allows the specification of time constraints (e.g. interval of time and duration) on sequence diagrams. Howbeit, the way to apply it on activity diagrams is not specified. Consequently, we propose in this paper an appropriate and straightforward time annotation, similar to the simple time model. The proposed notation allows us to specify duration variability with respect to action termination.

In this paper we address performance analysis of systems design described as SysML activity diagrams, using the Probabilistic Symbolic Model checker (PRISM) [12] developed at the University of Birmingham. PRISM addresses the assessment of probabilistic systems such as probabilistic communication protocols, security related systems, etc. In the reminder of this paper we discuss the related work in Section 2 and present the proposed SysML activity diagram time annotation in Section 3. Thereafter, Section 4 is dedicated to the description of our approach. Furthermore, we present in Section 5 the mapping procedure from a given activity diagram to its corresponding DTMC model. Subsequently, we present a real-life SE case study and show the assessment results in Section 6. Finally, we conclude with some summarizing remarks and future work.

2 Related Work

Presently, there are no initiatives on the verification and performance analysis of SysML models. This work aims at analyzing SysML activity diagrams annotated with time constraints in order to carry on requirement verification and performance analysis. While there are plenty of initiatives on verification as well as performance analysis of UML 1.x diagrams that essentially target software engineering, they are less relevant with respect to the scope of our work. Nevertheless, as SysML has its roots in UML 2.x, it is worthwhile to review some of the related work.

In [3], the authors are investigating software performance for UML 1.x models in conjunction with the SPT profile [9] that is used to annotate use case, activity, and deployment diagrams. The approach is based on deriving Queuing Networks models from the annotated diagrams. These models are used for performance prediction at the software architecture level. However, this work is suitable for software engineering rather than systems engineering. Trowitzsch et al. present in [15] the derivation of Stochastic Petri Nets (SPNs) from UML models for the purpose of Real-Time Systems (RTS) performance evaluation. The TimeNET (Timed Net Evaluation Tool) tool is proposed to support the quantitative analysis. The SPT UML profile is also considered for the design of RTS focusing on annotated real-time state machine diagrams and using an example to detail the proposed methodology. In [6], Canevet et al. propose the analysis of UML 2.0 activity diagrams in the context of software performance analysis using the Performance Evaluation Process Algebra (PEPA) net models. The authors are detailing the mapping from UML activity diagrams to their corresponding PEPA net models. In order to apply performance analysis, these models are used to generate the corresponding continuous-time Markov chain (CTMC) model wherein an exponential delay is associated with each corresponding activity node.

An interval based annotation for UML 1.x activity diagram has been proposed in [16]. However, in contrast to this work, we express time execution constraints for the termination of the actions within the annotated intervals according to a probability distribution.

3 Time-Annotated SysML Activity Diagrams

In this section we present SysML activity diagrams with a particular emphasis on the probability extension as specified by the standard. Furthermore, we describe the considered time constraint annotation.

SysML activity diagram is meant to describe control flow and dataflow dependencies among the functions/processes defined by the system. Figure 1 shows an example of a SysML activity diagram for a digital camera system that we consider as case study. SysML extends activity diagrams with probabilities. This feature can be used in two ways: on edges outgoing from decision nodes and as an extension to output parameter sets (the set of outgoing edges that hold data output from an action node) [10]. According to SysML specification, probabilities on edges express the likelihood that a value stemming from a decision node or object node will traverse an edge, whereas probabilities on output parameter sets express the likelihood that values will be output on a parameter set. In this paper, we only consider probabilities on edges leaving a decision node. Figure 1 illustrates the probability notation on outgoing edges from the decision node testing the guard `charged`. Its semantics states that there is a likelihood of 0.7 that the outcome of the decision node will be ($\texttt{charged} = \texttt{false}$) and the corresponding edge is traversed. The probabilities of the transitions emanating from the same decision node have to sum up to unity.

Generally, the execution time of a behavior depends on different parameters such as resource availability and rates of incoming dataflows. This may lead to a variable termination time of a behavior. Thus, we use a time annotation that shows this feature on the actions in the activity diagram. For

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

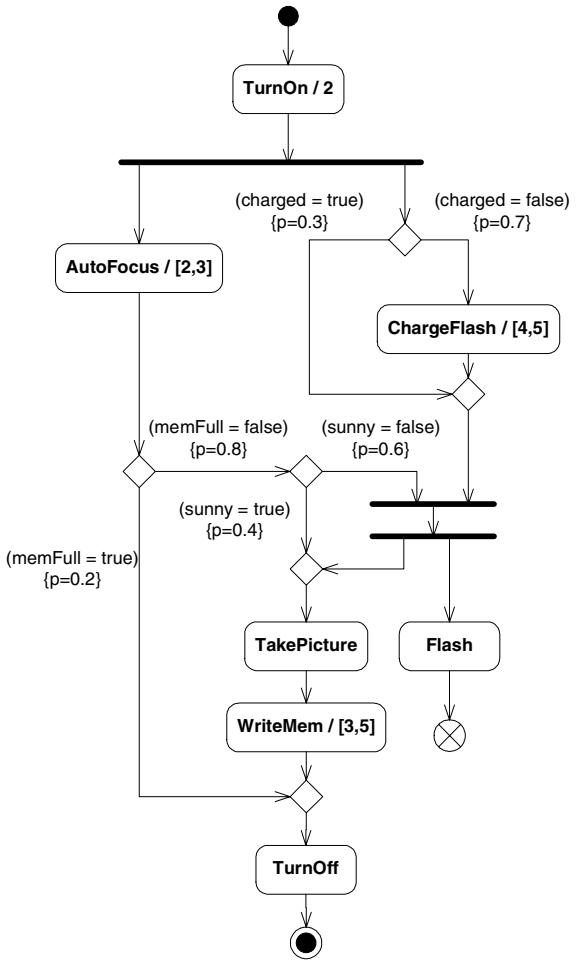


Figure 1. Digital Camera Activity Diagram

instance, an action may terminate within a bounded time interval. Hence, a probability distribution for terminating an action can be established with respect to the corresponding execution time interval. Consequently, we may have different time termination values according to the chosen probability distribution. Moreover, from the performance analysis point of view, the assessment of a model with respect to the time duration of its critical activities represents a strong point of interest. In the following, we propose a time annotation on top of the action nodes that is suitable for discrete-time modeling.

We denote by $TimeExp \subset \mathbb{N}$ the set of value specifications of time, where each value is expressed as a multiple of a time unit and $\mathbb{N}$ is the set of natural numbers. We consider a time reference maintained by a global clock $\mathcal{C}$ on which all the time values are evaluated. When an activity diagram starts, the clock $\mathcal{C}$ is reset to zero value. We denote by *activation time* the duration of time wherein the action

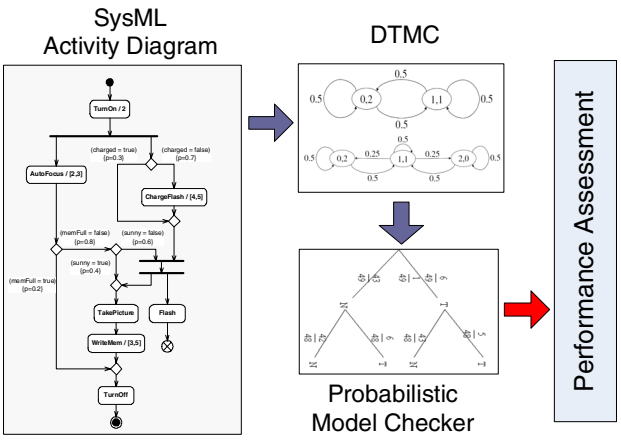


Figure 2. Proposed Approach

node is active. We consider also that a transition taken on an activity edge is timeless. Thus, an action node is annotated with a time interval $I = [a, b]$, where $a, b \in TimeExp$ are evaluated relatively to the start of the activation time of the corresponding action. Time value a represents the earliest time for the execution completion and b is the latest. However, some actions may need a fixed time value to complete execution, i.e. $a = b$ and in that case, only a time value is shown. Some actions may not have time annotation in the case where their activation time is negligible compared to other actions. As depicted in Figure 1, the action `TurnON` requires exactly 2 units of time to terminate; Action `AutoFocus` terminates within the interval [2, 3]; The action `TakePicture` execution time is negligible.

For convenience, we assume that there is an equal probability that an action terminates anytime within the interval I. At this point, although we consider hereafter a discrete equiprobable distribution, this has no restrictive impact with respect to our approach.

4 Approach

We present in this section our approach for the automatic verification and performance analysis of SysML activity diagrams that we extended with time constraints. Figure 2 illustrates the underlying procedure.

The approach consists in mapping the studied activity diagram to its corresponding DTMC representation. Then, the resulting model is encoded into the Probabilistic Symbolic Model Checker (PRISM) input language [12]. PRISM supports three types of probabilistic models: Discrete-Time Markov Chains (DTMC), Continuous-Time Markov Chains (CTMC), and Markov Decision Process (MDP). Since our main objective is to capture the dynamics of activity diagrams into a compact computable model, we found that the

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

DTMC model provides a suitable semantic interpretation for this type of diagram because it can model coordination of behavior, actions execution duration, synchronizations as well as probabilistic path selection as captured by activity diagrams. Moreover, DTMC is a discrete-time transition system with discrete probability distributions that is lightweight compared to the other probabilistic models supported by PRISM. Relating to the other probabilistic models supported by PRISM, CTMC can be conceived as DTMC with an infinitesimally small time step, whereas MDP is extending DTMC with non-determinism [12]. Formally, according to [14], a DTMC is a tuple $\mathcal{D} = (S, \overline{s}, \mathbf{P})$ where S is a finite set of states, $\overline{s} \in S$ is the initial state, and $\mathbf{P} : S \times S \rightarrow [0, 1]$ is a transition probability matrix, such that: $\sum_{s' \in \mathcal{S}} \mathcal{P}(s, s') = 1$ for all $s \in S$ where $\mathcal{P}(s, s')$ is the probability of making a transition from a state s to a state s'.

We employ the capability of PRISM to determine the actual value of the probability for a given behavior occurrence, probabilities for best/worst cases as well as probabilities with parameterized ranges of model properties and accordingly perform quantitative analysis on the design model. Moreover, since the counter example generation feature is not supported yet, we use the state space computed by PRISM to construct the reachability graph of the model in order to graphically explore its behavior.

5 Mapping SysML Activity to DTMC

PRISM language is based on reactive modules [1], where each module represents a system component. The modules are composed in parallel and contain variables as well as commands labeled with actions for synchronization between modules, hidden actions, guards and probabilities. The first step in transforming SysML activity to DTMC consists in identifying different communicating modules corresponding to the synchronizing threads in the diagram.

In order to be able to delimit the existing individual threads, we have to explore the different flows of the activity diagram. Forks and joins are special constructs that represent respectively spawning and synchronization points. In fact, at each *fork*, there are as many new threads as outgoing edges and at each *join*, there is a waiting point for all the incoming threads to synchronize. Accordingly, we allocate a module for each identified thread. However, there are special cases where some paths are intersecting with each other. In that case, we chose to merge the corresponding modules. Also, when a thread reaches a join construct, it synchronizes with others and terminates, the corresponding module can be reused to contain another subsequent thread. This is done for the sake of optimizing the PRISM code in order to keep it easily manageable. Algorithm 1 presents the generic procedure for converting an activity diagram into its corresponding DTMC. The synchronization mechanism

Algorithm 1 Mapping Activity Diagram *AD* to PRISM DTMC

```
CTT = GetStructure(AD) {Encodes the structural artifacts into Control Transfer Table (CTT)}
FEP = generateFEP(CTT) {Explores CTT and generates the list of Flow Execution Paths (FEP)}
SSP = GetSynchropoints(CTT,FEP) {Generates a list of States Synchronization POINTs (SSP)}
for all item in FEP do
    LCT= Compact(FEP) {Compacts the overlapping parts of FEP into LCT (list of connecting threads)}
end for
for all item in LCT do
    CreateNEWModule(item) {Encodes each thread in LCT along with its SSP as a separate PRISM module}
    PRISMModulesList.add(newPRISM) {Generates PRISM code}
end for
```

among the modules allows for two or more concurrent activity flows or execution threads to "experience" the same passage of time with respect to the time constraints that may be specified for each of them. Consequently, each thread has its own clock variable that is used to track the passage of time in the current state of the thread. The dynamics of the model ensures that all the clock variables are updated (advanced or reset) synchronously. Thus, the clock variable of each thread is advanced as a result of a self transition to its current state or reset whenever the current state is left and another one is entered. Furthermore, whenever the clock variable of a thread falls within the time constraint interval of the current state, this means that the control can either remain in the current state or be transferred to another state that can be reached by a transition from the current state. This amounts to the use of a probability distribution based on which the thread will remain in the current state or exit it. Though the choice for such a distribution may depend on the actual system being modeled, by default we use a uniform distribution over the time constraint interval.

6 Case Study

In this section, we present a relevant case study involving a systems engineering product for which a SysML activity diagram with time constraints is analyzed. The system represents a hypothetical model of a digital photo-camera while the given activity diagram is meant to capture the functional aspect of taking a picture as depicted by Figure 1. Furthermore, the presented activity diagram is intentionally not very laborious as to serve its didactic purpose. However, as it will become apparent later, a simple model does not preclude a highly dynamic behavior. Later, we present

properties that aim to measure and verify certain performance characteristics exhibited by the activity diagram.

For our example, the selection of an appropriate unit of time from the sequencing and performance perspectives has to be relevant for the user of the product. Accordingly, the activity diagram nodes that may require one or more units of time to complete are augmented with a corresponding value of interval that denotes the expected time to completion. In contrast, the activity nodes that take less than one unit of time to complete have no such augmentation.

The process captured by our activity diagram example starts by turning on the camera (`TurnOn`). Subsequently, two threads are spawned. The first one commences by auto focusing (`AutoFocus`) followed by a decision point that checks whether the memory is full (`memFull` guard) or not. In the latter case (`memFull = false`), a second decision point is reached, where depending on the ambient lighting conditions (sunny or not) taking a picture (`TakePicture`) is executed. `TakePicture` may be reached either directly (`sunny = true`) or after synchronizing with the second thread (`sunny = false`). Thereafter, the picture is stored in the memory (`WriteMem`). The second thread begins by checking whether the flash is charged or not. If not, the control is transferred to charging the flash (`ChargeFlash`). Subsequently or if the flash is already charged, the second thread awaits for the first thread in order to synchronize (join node). After synchronization, the flashing is triggered (`Flash`). The activity diagram execution is ended with turning off the camera (`TurnOff`).

To assess the activity diagram of the digital camera, we generated the corresponding DTMC according to the procedure depicted in Algorithm 1 and encoded it into PRISM model checker input language as shown in Figure 3. The corresponding reachability graph, derived from the state and transition matrices computed by PRISM version 3.0, is presented in Figure 4. As illustrated, each edge of the reachability graph is annotated with its corresponding probability of selection. Furthermore, the graph shows a highly dynamic behavior resulting from the concurrency of the threads in the activity diagram in conjunction with the overlapping completion intervals of various activity nodes.

For the specification of the properties involved in the quantitative analysis, PRISM incorporates two property specification languages: Probabilistic Computation Tree Logic (PCTL) and Continuous Stochastic Logic (CSL). PCTL is used with DTMC and MDP models, whereas CSL is used with CTMC. Since we are interested in DTMC model, we briefly remind the PCTL syntax. PCTL [7] is an extension of CTL [8] mainly with probability operator. According to [7] its syntax is as follows:

$$\begin{aligned} \phi &::= true \mid a \mid \neg\,\phi \mid \phi \wedge \phi \mid \mathcal{P}_{\bowtie p}[\psi] \\ \psi &::= \phi_1\,\mathcal{U}^t\,\phi_2 \mid \phi_1\,\mathcal{U}\,\phi_2 \mid \mathcal{X}\phi \end{aligned}$$

where a is an atomic proposition, $t \in \mathbb{N}$, $p \in [0,1] \subset \mathbb{R}$, and $\bowtie \in \{>, \geq, <, \leq\}$. PCTL formulas are of two types, state (ϕ) and path (ψ). The former is evaluated over a state and the latter over a path. For example, a state s of the DTMC satisfies the formula $\mathcal{P}_{\bowtie p}[\psi]$ if the probability of taking a path from s satisfying ψ is in the interval defined by $\bowtie p$.

In the following, we present the results obtained by verifying a number of interesting properties expressed in the syntax of the temporal logic defined by PRISM and based on PCTL. This highlights the benefits of assessing probabilistic behavior in the presence of time constraints.

The first property (1) states that the probability to take a picture after turning on the camera should be 0.75 or greater. It is expressed in PRISM as follows:

$$\texttt{TurnOn} \Rightarrow \texttt{P} \geq 0.75\,[\,\texttt{true}\ \texttt{U}\ \texttt{TakePicture}\,] \qquad (1)$$

When analyzing this property over the DTMC of the activity diagram, PRISM reported that it turns out to be satisfied. This reflects that the model meets the minimum required level of reliability. The second property (2) is building on the previous one and aims to check whether in the case of poor lighting conditions, the probability of taking a picture using the flash after turning on the camera is at least 0.5. It is expressed in PRISM as follows:

$$\begin{aligned} \texttt{TurnOn} \Rightarrow \texttt{P} \geq 0.5\,[\,\texttt{true}\ \texttt{U}\ !\texttt{sunny}\ \& \\ \texttt{TakePicture}\ \&\ \texttt{Flash}\,] \end{aligned} \qquad (2)$$

Though the model checker determined that the property fails for a probability value greater than or equal to 0.5, it passes for a value less than or equal to 0.48. The actual probability can be determined by using a specific PRISM construct that queries for the probability value (3). This illustrates the feature of probability quantitative measurement provided by PRISM. The formula is as follows:

$$\begin{aligned} \texttt{P} = ?\,[\,\texttt{true}\ \texttt{U}\ !\texttt{sunny}\ \&\ \texttt{TakePicture}\ \& \\ \texttt{Flash}\ \{\texttt{TurnOn}\}\,] \end{aligned} \qquad (3)$$

Apart from the foregoing properties that concern the probability of reaching a particular situation, we also consider time-bounded reachability property verification which consists in measuring the probability of reaching a certain situation within a time bound. For instance, we have the property (4) measuring the probability of taking a picture using the flash within 6 units of time after turning on the camera. One can note that 2 supplementary units of time are added in the property formula. The reason behind this is due to the fact that when reaching a synchronization point (join node), each thread needs to enter a waiting state wherein it awaits for the other synchronizing threads to complete their respective activities. When all the synchronizing threads reach the synchronization point, they require one unit of

```
probabilistic

module t1
        TurnOn: bool init true;
        t1_e: bool;
        t1_ck: [0..2] init 0;
        [step] TurnOn & t1_ck<2 -> t1_ck'=t1_ck+1;
        [step] TurnOn & t1_ck=2 -> t1_ck'=0 & TurnOn'=false & t1_e'=true;
        [step] t1_e -> t1_e'=t1_e;
endmodule

formula t2_ca = AutoFocus | TakePicture | WriteMem | TurnOff | t2_j | t2_e;
module t2
        AutoFocus: bool;
        TakePicture: bool;
        WriteMem: bool;
        TurnOff: bool;
        memFull: bool;
        sunny: bool;
        t2_j: bool;
        t2_e: bool;
        t2_ck: [0..5] init 0;
        [step] !t1_e & !t2_ca -> t2_e'=t2_e;
        [step] t1_e & !t2_ca -> AutoFocus'=true;
        [step] AutoFocus & t2_ck < 2 -> t2_ck'=t2_ck+1;
        [step] AutoFocus & t2_ck >=2 & t2_ck < 3 -> 0.5: (t2_ck'=t2_ck+1) +
                        0.5*0.2:     (t2_ck'=0) & (AutoFocus'=false) & (TurnOff'=true) & (memFull'=true) +
                        0.5*0.8*0.4: (t2_ck'=0) & (AutoFocus'=false) & (TakePicture'=true) & (sunny'=true) +
                        0.5*0.8*0.6: (t2_ck'=0) & (AutoFocus'=false) & (t2_j'=true);
        [step] AutoFocus & t2_ck >=3 ->
                        0.2:     (t2_ck'=0) & (AutoFocus'=false) & (TurnOff'=true) & (memFull'=true) +
                        0.8*0.4: (t2_ck'=0) & (AutoFocus'=false) & (TakePicture'=true) & (sunny'=true) +
                        0.8*0.6: (t2_ck'=0) & (AutoFocus'=false) & (t2_j'=true);
        [step] t2_j & !t3_j -> t2_j'=t2_j;
        [step] t2_j & t3_j -> t2_j'=false & TakePicture'=true;
        [step] TakePicture -> TakePicture'=false & WriteMem'=true;
        [step] WriteMem & t2_ck < 3 -> t2_ck'=t2_ck+1;
        [step] WriteMem & t2_ck >=3 & t2_ck < 5 -> 0.5: (t2_ck'=t2_ck+1) +
                                        0.5: (t2_ck'=0) & (WriteMem'=false) & (TurnOff'=true);
        [step] WriteMem & t2_ck >=5 -> (t2_ck'=0) & WriteMem'=false & TurnOff'=true;
        [step] TurnOff -> TurnOff'=false & t2_e'=true;
        [step] t2_e -> t2_e'=t2_e;
endmodule

formula t3_ca = ChargeFlash | Flash | t3_j | t3_e;
module t3
        ChargeFlash: bool;
        Flash: bool;
        charged: bool;
        t3_j: bool;
        t3_e: bool;
        t3_ck: [0..5] init 0;
        [step] !t1_e & !t3_ca -> t3_e'=t3_e;
        [step] t1_e & !t3_ca -> 0.3: (charged'=true) & (t3_j'=true) +
                        0.7: ChargeFlash'=true;
        [step] ChargeFlash & t3_ck < 4 -> t3_ck'=t3_ck+1;
        [step] ChargeFlash & t3_ck >=4 & t3_ck < 5 -> 0.5: (t3_ck'=t3_ck+1) +
                                        0.5: (t3_ck'=0) & ChargeFlash'=false & t3_j'=true;
        [step] ChargeFlash & t3_ck >=5 -> (t3_ck'=0) & ChargeFlash'=false & t3_j'=true;
        [step] t3_j & !t2_j -> t3_j'=t3_j;
        [step] t3_j & t2_j -> t3_j'=false & Flash'=true & charged'=false;
        [step] Flash -> Flash'=false & t3_e'=true;
        [step] t3_e -> t3_e'=t3_e;
endmodule
```

Figure 3. PRISM code for the Digital Camera Activity Diagram Example

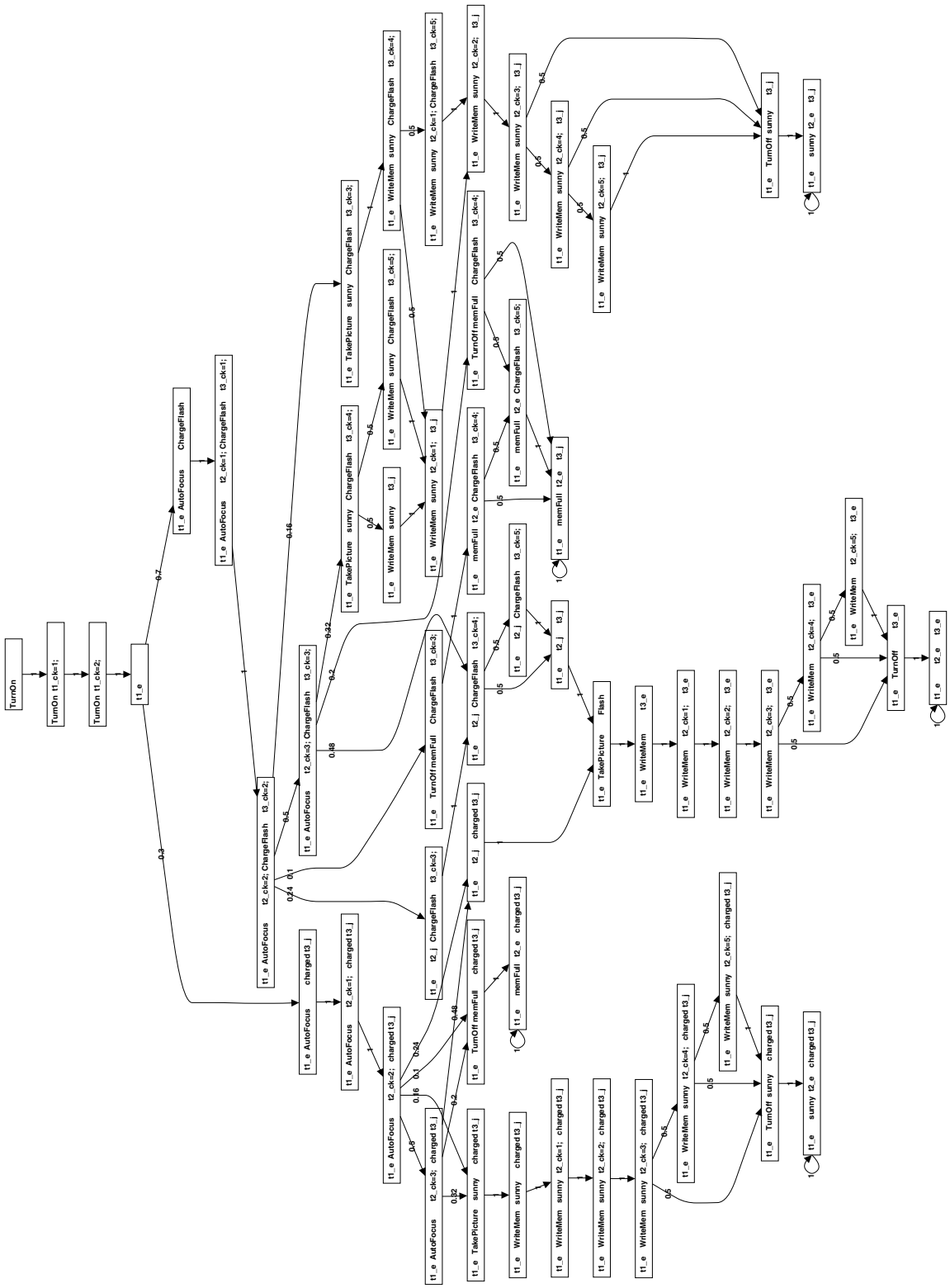


Figure 4. Reachability Graph of the Digital Camera Activity Diagram Example

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

time to leave their waiting states and one unit of time to perform the synchronization and proceed further. Thus, for this kind of properties involving synchronizing threads, one has to take this fact into account. In our example depicted in Figure 3, only modules t2 and t3 are synchronizing.

$$\mathtt{P} = ?\,[\,\mathtt{true}\ \mathtt{U}\ \leq 6 + 2\ \mathtt{TakePicture}\ \&\ \mathtt{Flash}\ \{\mathtt{TurnOn}\ \&\ \mathtt{t1\_ck} = 0\}\,] \quad (4)$$

The corresponding results indicate a severe performance issue for the considered time constraint requirement since the computed value of the probability is 0.072. This means that the likelihood to take a picture within 6 units of time is less than $1/10$. Nevertheless, considering the fact that charging the flash may take a significant amount of the 6 unit of time, the obtained result reflects either a bad design or that this particular performance requirement is not mandatory but rather optional. However, the model checker has determined higher probability values for more relaxed time constraints. Thus, a value of 0.144 was identified for a time interval of 7 units, 0.312 for 8 units and 0.48 for 9 units.

7 Conclusion and Future Work

In this paper, we presented a new approach for the automatic verification and performance analysis of SysML activity diagrams. We introduced time constraint annotation to enable the specification of execution duration on actions and support the performance assessment of the model. The proposed methodology is based on mapping SysML activity diagrams to their corresponding DTMC models according to a transformation procedure that we devised. Then, the resulting DTMC is encoded into the PRISM model checker input language for qualitative and quantitative analysis. To demonstrate the feasibility of our methodology, we also presented a relevant case study concerning the performance assessment of a SE design capturing the functional aspect in a SysML activity diagram annotated with time constraints and probability artifacts.

As future work, we intend to further investigate the presented procedure for mapping activity diagrams to DTMC models in order to determine the critical time-constrained path. Moreover, this can support an automated back annotation of the analyzed SysML activity diagram as a performance assessment feedback. Furthermore, the proposed approach may be extended to handle multiple activity diagrams. This might be done by a preliminary step wherein the diagrams to be analyzed will be composed into a single one according to their dependence.

References

[1] R. Alur and T. A. Henzinger. Reactive Modules. *Form. Methods Syst. Des.*, 15(1):7–48, 1999.

[2] S. Balsamo, A. D. Marco, P. Inverardi, and M. Simenoni. Model-Based Performance Prediction in Software Development: A Survey. *IEEE Transactions on Software Engineering (TSE)*, 30(5):295–310, May 2004.

[3] S. Balsamo and M. Marzolla. Performance Evaluation of UML Software Architectures with Multiclass Queueing Network Models. In *WOSP '05: Proceedings of the 5th international workshop on Software and performance*, pages 37–42, New York, USA, 2005. ACM Press.

[4] C. Bock. Systems Engineering in the Product Lifecycle. *Int. Journal Product Development*, 2:123137, 2005.

[5] C. Bock. SysML and UML 2 support for Activity Modeling. *Syst. Eng.*, 9(2):160–186, 2006.

[6] C. Canevet, S. Gilmore, J. Hillston, L. Kloul, and P. Stevens. Analysing UML 2.0 Activity Diagrams in the Software Performance Engineering Process. In *WOSP '04: Proceedings of the 4th international workshop on Software and performance*, pages 74–78, New York, USA, 2004. ACM Press.

[7] F. Ciesinski and M. Größer. On Probabilistic Computation Tree Logic. In *Validation of Stochastic Systems*, pages 147–188, 2004.

[8] E. M. Clarke and E. A. Emerson. Design and Synthesis of Synchronization Skeletons Using Branching-Time Temporal Logic. In *Logic of Programs, Workshop*, pages 52–71, London, UK, 1982. Springer-Verlag.

[9] O. M. G. *UML Profile for Schedulability, Performance and Time*, January 2005.

[10] O. M. G. *OMG Systems Modeling Language (OMG SysML) Specification*, May 2006. Final Adopted Specification of Systems Modeling Language (SysML).

[11] O. M. G. *Unified Modeling Language: Superstructure Version 2.1*, April 2006.

[12] M. Kwiatkowska, G. Norman, and D. Parker. Quantitative Analysis with the Probabilistic Model Checker PRISM. *Electronic Notes in Theoretical Computer Science*, 153(2):5–31, 2005.

[13] I. C. on Systems Engineering (INCOSE). INCOSE 2003 Fact Sheet. `http://www.incose.org/symp2003/Sponsor%20Fact%20sheet.pdf`.

[14] J. Rutten, M. Kwiatkowska, G. Norman, and D. Parker. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*, volume 23 of *CRM Monograph Series*. American Mathematical Society, 2004.

[15] J. Trowitzsch, A. Zimmermann, and G. Hommel. Towards Quantitative Analysis of Real-Time UML Using Stochastic Petri Nets. In *IPDPS '05: Proceedings of the 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS'05) - Workshop 2*, page 139.2, Washington, DC, USA, 2005. IEEE Computer Society.

[16] L. Xuandong, C. Meng, P. Yu, Z. Jianhua, and Z. Guoliang. Timing Analysis of UML Activity Diagrams. In *UML'01: Proceedings of the 4th Int. Conf. on The Unified Modeling Language, Modeling Languages, Concepts, and Tools*, pages 62–75, London, UK, 2001. Springer-Verlag.

System Level Performance Assessment of SOC Processors with SystemC

Claudio Talarico[†], Min-sung Koh[†], Esteban Rodriguez-Marek[†]

[†]School of Computing and Engineering Sciences,
Electrical Engineering
Eastern Washington University
Cheney, WA 99004, USA

Abstract

This paper presents a system level methodology for modeling, and analyzing the performance of system-on-chip (SOC) processors. The solution adopted focuses on minimizing assessment time by modeling processors behavior only in terms of the performance metrics of interest. Formally, the desired behavior is captured through a C/C++ executable model, which uses finite state machines (FSM) as the underlying model of computation (MOC). To illustrate and validate our methodology we applied it to the design of a 16-bit reduced instruction set (RISC) processor. The performance metrics used to assess the quality of the design considered are power consumption and execution time. However, the methodology can be extended to any performance metric. The results obtained demonstrate the robustness of the proposed method both in terms of assessment time and accuracy.

1. Introduction

In recent years, the increasing complexity of computing system applications, the very large scale of integration (VLSI) of transistors on silicon, and the consistent strive to reduce time-to-market and costs have resulted in a substantial change in the way that systems are designed, analyzed, and verified. In this context, system-on-chip (SOC) designs have become one of industry's most common practices [1]. SOC designs are composed by a collection of heterogeneous components such as processors, memory hierarchy, communication buses, application specific hardware functions, peripheral control devices, and millions of lines of software [2],[3]. Often the software design effort associated with an SOC is larger than the hardware design effort [4].

The work presented in this paper addresses the problem of assessing SOC processors and the associated application software by introducing an estimation technique in which power consumption and execution time are obtained through the execution of high level abstraction models. The distinguishing benefits of using executable models that support high-level abstract descriptions are: 1) performance estimation can be performed much earlier in the design process (i.e., when design decisions have the biggest impact on performance), and 2) time required to explore different design tradeoffs is much shorter.

Raising the level of abstraction (i.e., reducing the level of detail used to model a system) is widely seen as the only viable solution to cope with the complexity of today's SOC designs [5]. The current trend is for designers to model systems using different levels of abstraction depending on the design step being considered. Since, raising the level of abstraction implies trading off accuracy for efficiency, as design evolves the system description is usually refined with more accurate and detailed information. This allows for different design concerns to be addressed and solved and design reuse facilitated 0.

The rest of this paper is organized as follows. Section 2 introduces the basic tenets on which our methodology is based. Section 3 discusses the general design and implementation of our methodology. Section 4 demonstrates the viability of our approach by applying it to a design example consisting of a 16-bit processor produced by Infineon Technologies [6]. Section 5 discusses the results obtained and compares them with direct physical measurements. Finally, Section 6 summarizes the results of our work and provides conclusions.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

2. Background

Extensive research has been conducted on the problem of assessing SOC processors performance and several interesting solutions have been proposed. In general, designers can estimate the performance of any system at four different abstraction levels [8]:

- *Circuit-level* approaches simulate the design at the transistor or switch level. The execution time is too high. It would require weeks to simulate an average SOC processor.
- *Logic-level* techniques simulate the design at the logic-gate level. Logic-level approaches execute orders of magnitude faster than circuit-level approaches but at the expense of accuracy.
- *Register-transfer-level* approaches simulate the design at the functional level using Hardware Description Languages (HDL). They have satisfactory accuracy (5-10 percent of gate level estimates), but their computational time, while orders of magnitude smaller than logic-level approaches, is still too slow when applied to large designs.
- *System-level* approaches estimate performance based on simple high-level descriptions of the system's behavior and its intended application.

Different estimation techniques are best suited to different parts of a design or different stages in the design cycle. For example, at the early stage of the design cycle, the detailed internal structure knowledge required for modeling a processor at the circuit-level, logic-level, or register-transfer-level (RTL) may not be available at all, or intellectual properties (IP) providers may not want to disclose it.

Designing an SOC processor requires many capabilities: 1) describing the interaction between the design and the external environment, 2) describing the design architecture, 3) modeling the behavior of hardware and software components forming the design, 4) describing design constraints and requirements, 5) describing the test scenarios used to simulate the design, and 6) defining a set of gauges to measure the performance metrics of interest during simulation execution. In order to capture all these aspects of a system behavior we can rely on many different mathematical MOC. Some examples of commonly used MOC include finite state machines, data flow process networks, discrete event models, and Petri nets [9][10]. Since a MOC is just a conceptual notion, we need a language to capture that concept in a concrete form. In practice certain languages may be better than other at capturing different MOC, thus the complexity of modeling a design is also determined by the semantic and syntax of the language adopted. In general, a system-level-design language requires two essential attributes: 1) it should support modeling at all levels of abstraction, from purely behavioral un-timed models to cycle accurate RTL models, and 2) the models should be executable and simulatable, so that functionality, performance and constraints can be validated [11].

In our approach we decided to use finite state machines as MOC and SystemC as implementation platform. Formally, a finite state machine is a 6-uple (I, X, O, ξ, ω, x_0), where:

I is a finite set of inputs: $I = \{i_1, i_2, ..., i_{|I|}\}$

X is a finite set of internal states $X = \{x_1, x_2, ..., x_{|X|}\}$

O is a finite set of outputs $O = \{o_1, o_2, ..., o_{|O|}\}$

ξ is a next state transition function $\xi : I \times X \rightarrow X$

ω is an output function $\omega : I \times X \rightarrow O$

x_0 is the initial state, $x_0 \in X$

SystemC is a modeling platform based on C++, and it consists of a C++ compiler, C++ class libraries and a simulation kernel. SystemC provides a unified framework for modeling both hardware and C/C++ software. The class libraries support hardware modeling at different abstraction levels including un-timed functional level, transaction level, and cycle-accurate register transfer level. Besides, providing a common high-level language, for modeling, analyzing and simulating a design, SystemC can be also linked to commercial tools for hardware synthesis, such as Synopsys' design compiler [12]. Detailed information on the syntax and semantics of SystemC are available in reference [13].

3. Performance Assessment Framework

The objective of this work is to develop a framework to assess power consumption and execution time of SOC processors. Given the diversity

of today's applications, searching for a processor that optimally satisfies the application's requirements and constraints it involves exploring a very large number of design alternatives. To evaluate performance, we cannot afford to synthesize and simulate at the cycle level every possible design alternative. It is imperative to estimate performance as rapidly as possible, and as early as possible in the design cycle. In order to address this problem we propose an estimation technique that derives time and power consumption from the execution of high level models rather than relying on gate- or transistor-level characterizations. To that end, the use of a high level language (e.g., C/C++) not only simplifies the modeling task, but it also maintains computation time within feasible ranges [14].

Given an application program execution trace, the approach proposed computes the energy and time that each executed instruction consumes. Both time and energy consumption depends on the specific instruction being executed as well as on previously executed instructions and on the data on which the instruction operates. Figure 1 illustrates the basic structure of our framework for estimating execution time and power consumption of an SOC processor.

The framework functions as a wrapper around the processor modeling elements. Each processor has associated a simulation model, a monitor, and an analyzer. The monitor observes the model's execution and probes the data needed to characterize the component's behavior. The analyzer then computes the performance indices of interest. Our framework generalizes and extends to execution time the scheme developed in [8] and [15]. In our view, a component's behavior can be seen as the execution of a sequence of instructions, in which the term "instruction" is synonymous with "action" and it is not necessarily atomic, therefore it does not necessarily map with one of the processor instructions.

The framework consists of four steps that lead to an estimate of power consumption and execution time:

- *translatin*g processor's functionality to a set of primitive instructions,
- *simulating* the application program,
- *mapping* the instructions requested by the application program into abstract functional units, and
- *computin*g aggregate power consumption and execution time

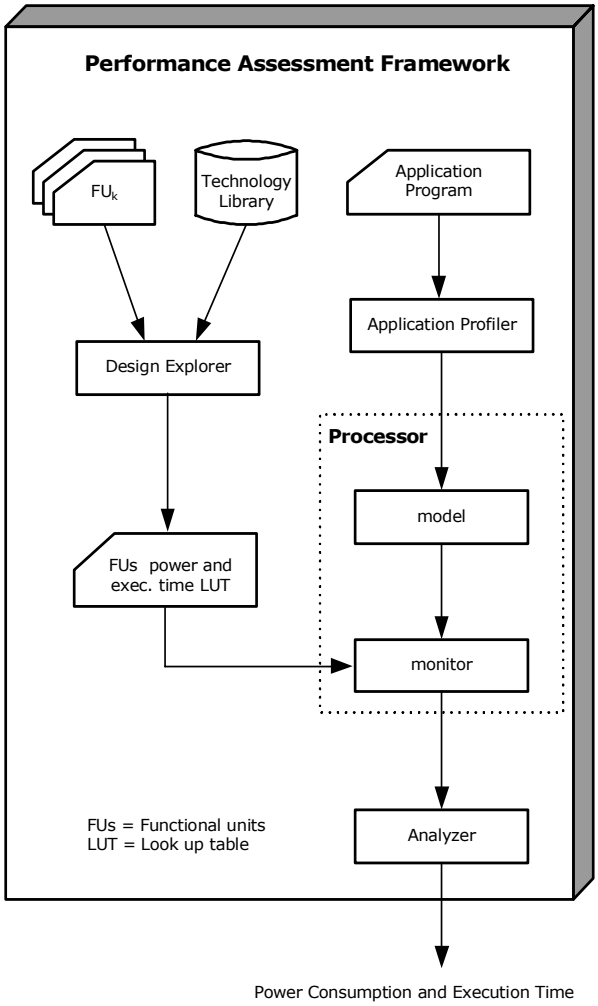


Fig. 1. Performance Assessment Framework for SOC Processors

The first step consists of breaking the processor's functionality into a set of instructions. A component's functionality represents all possible behaviors that the components can assume, with behavior meaning the set of actions that the component performs during the execution of an application. The goal is to devise a high level executable model of the core that can output power consumption and execution time information during system simulation. This first step hides the complexity of the processor's internal implementation behind the simple interface offered by the instruction set. There is a tradeoff in selecting the right set of instructions: having many fine-grained

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

instructions can lead to greater accuracy, but it requires a longer simulation time than having fewer coarse-grained instructions.

The second step involves simulating the application program and extracting a trace file for the processor. A trace is the sequence of instructions/data items a component executes during its simulation. The aim is to estimate the component's switching activity.

The third step consists of mapping the instructions requested by the various tasks performed by the component into abstract functional units that are used to estimate complexity—that is, gate count and timing delay. Given switching activity and complexity, the framework can compute the component's power per instruction and execution time per instruction.

The fourth step involves computing the power consumption and execution time for the entire application executed on the processor.

The performance analyzer module shown in Fig. 1 embodies the analytical expressions needed to compute power and execution time consumed by the processor.

The input to the estimation flow is the application program which feed into the application profiler for the target processor. The primary output of the application profiler is the program trace. In addition, the profiler maintains detailed statistics of the processor's internal activity (e.g., fetches, stalls, instruction execution frequency, and internal register accesses).

4. A Design Example

To illustrate and validate the proposed approach we applied it to the design of a 16-bit processor produced by Infineon Technologies. To compute execution time and power per cycle of each processor's instruction we employed Infineon's 0.25µm CMOS technology. We estimated the average capacitance and propagation delay of combinational cells and sequential cells separately, and stored the resulting values in a look-up table to facilitate access during model execution. The left-hand side of Fig. 1 shows the implementation details. The design explorer analyzes the functional units forming the processor, estimates their complexity (total capacitance and propagation delay) based on the target technology, and evaluates each unit's power consumption and execution time.

The processor operates at 40 MHz clock cycle, and it optimizes instruction throughput using a 5-stage execution pipeline architecture. As shown in Fig. 2, the processor consists of eight main functional units: an instruction fetch unit (IFU), an address and data generation unit (ADU), a 5-stage instruction execution pipeline (IPIP), an injection/exception handler, an arithmetic and logic unit (ALU), a multiply and accumulate unit (MAC), a register file (RF), and a write-back buffer (WB).

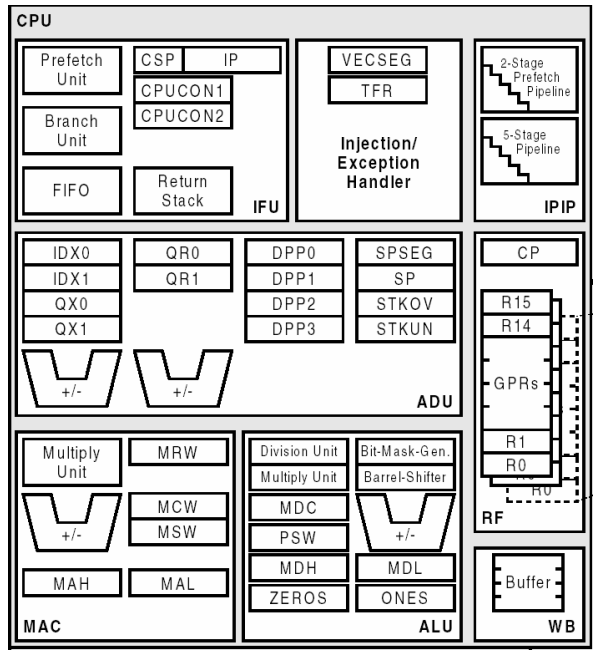


Fig. 2. Infineon's C166SV2 processor: main functional units

The modeling and simulation of the processor has been entirely implemented using SystemC. This choice allows the use of the same language (i.e., C/C++), for describing both hardware and software. The idea is to keep the hardware and software design activities tightly coupled 0. In order to model the behavior of the processor with respect to power consumption and execution time only a small amount of detail is required. The behavior is described by using the simple state machine given in Fig. 3. The use of state machines makes it easier to account for the dependency of consecutive operations. Power consumption and execution time have been evaluated on ten different application programs.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

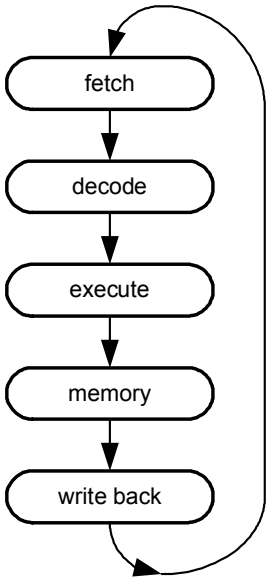


Fig. 3. State machine used for modeling the processor behavior

The application programs used are: insertionSort, bubbleSort.c, quickSort.c, binarySearch.c, fibonacci.c, sieve.c, eightQueens.c, pascalTriangle.c, euclid.c, and stringSearch.c

5. Results

In order to validate the accuracy of our proposed method, we simulated the execution on the processor of ten different application programs and compared the results obtained with those obtained taking direct measurement of execution time and absorbed current. Having defined performance as the product between power consumption and execution time we obtain that the accuracy of our method is within 10% and the average error is 7.64%. Compared to gate level estimation our system-level approach achieves a speedup of three orders of magnitude. Fig. 4 summarizes for each benchmark program the difference between the performance estimated and the performance measured. System-level estimation is consistently lower than gate-level estimation, because the former always considers a lower level of detail than the latter.

6. Conclusions and Future Work

In this paper we have presented a system level framework for assessing power consumption and execution time of SOC processors early in the design cycle. The approach is entirely implemented in C/C++ and it uses SystemC as platform. The accuracy of the approach has been evaluated by executing ten different application programs on the design of a 16-bit processor commercially produced by Infineon Technologies. Results show that the proposed method provides accuracy within 10% of actual physical measurements, and an average error of 7.64%. Compared to gate level simulation our approach achieves a speedup of three orders of magnitude. Although, the methodology focuses on power consumption and execution time, its generality makes it extendable to any performance metric. Future work includes testing the approach with more complex application programs and on different processors. The aim is to better evaluate the scalability of the approach.

6. References

[1] K. Keutzer, S. Malik, A.R. Newton, J.M. Rabaey, and A. Sangiovanni-Vincentelli, “System-Level Design: Orthogonalization of Concerns and Platform-Based Design,”, *IEEE Trans. CAD Integrated Circuits and Systems*, vol. 19, no. 12, 2000, pp. 1523-1543.

[2] T. Givargis, and F. Vahid, “Platune: A Tuning Framework for System-on-a-Chip Platforms,” *IEEE Trans. CAD Integrated Circuits and Systems*, vol. 21, no. 11, 2002, pp. 1317-1327.

[3] V. Tiwari, S. Malik, and A. Wolfe, “Power Analysis of Embedded Software: A First Step toward Software Power Minimization,” *IEEE Trans. VLSI Systems*, 1994, pp. 437-445.

[4] S. Swan “SystemC Transaction Level Models and RTL Verification,” *Proc. 43th Design Automation Conf.* (DAC 2006), IEEE Press, 2006, pp. 90-92.

[5] A. Gerstlauer and D.D. Gajski, “System-Level Abstraction Semantics,” *Int’l Symp. System Synthesis* (ISSS 2002), IEEE Press, 2002, pp. 231-236

.

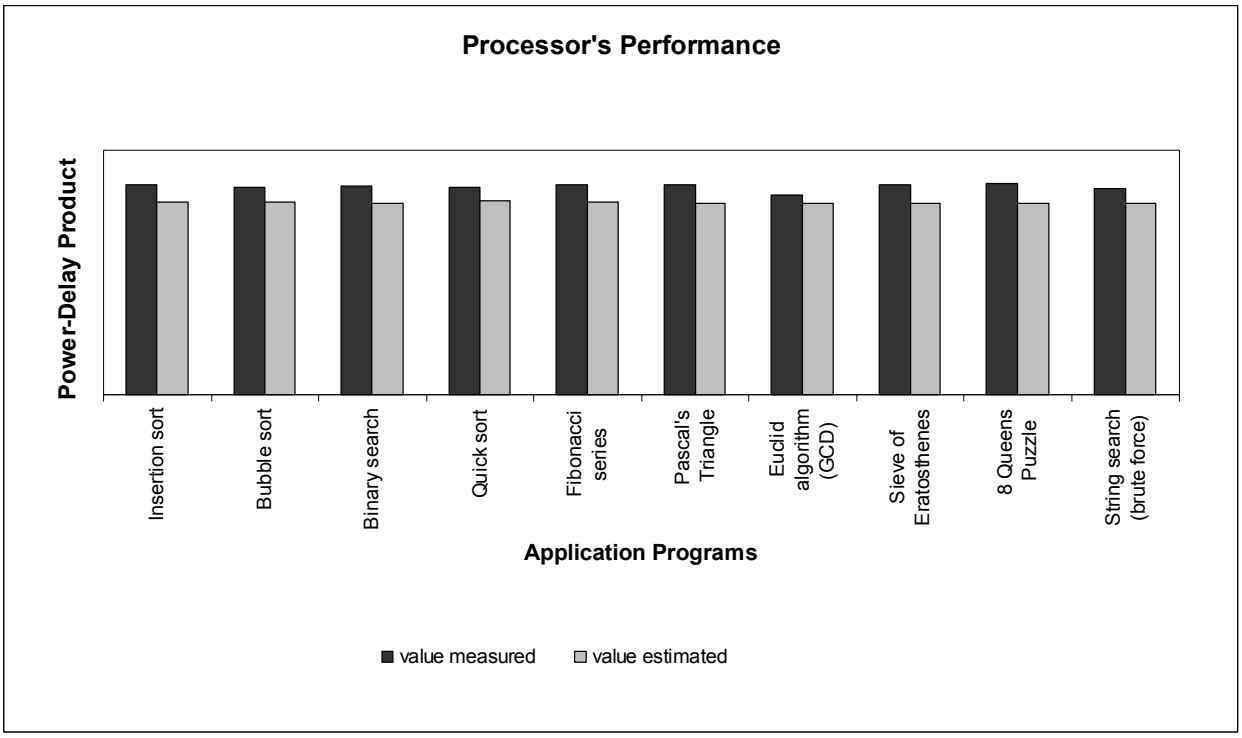


Fig. 4. Performance benchmarks: measured vs. estimated

[6] C. Talarico, A. Gupta, E. Peter, and J.W. Rozenblit, "Embedded System Engineering using C/C++ based Design Methodologies," *Proc. Int'l Conf. Engineering Computer-Based Systems* (ECBS 2005), IEEE Press, 2005, pp. 81-88.

[7] Infineon Technologies AG, "XC161, 16-Bit Single-Chip Microcontroller with C166SV2 Core: User's Manual v2.2," 2004; http://www.infineon.com

[8] J. Henkel and Y. Li, "Avalanche: an environment for design space exploration and optimization of low-power embedded systems," *IEEE Trans. VLSI Systems*, vol. 10, no. 4, 2002, pp. 454-468.

[9] M. Sgroi, L. Lavagno, A. Sangiovanni-Vincentelli, "Formal Models for Embedded System Design," *IEEE Design & Test of Computers*, 2000, pp. 2-15.

[10] A. Jantsch, *Modeling Embedded Systems and SOCs*, Morgan-Kaufmann Publishers, 2004.

[11] D.D. Gajiski, J. Zhu, R. Doemer, A. Gerstauler and S. Zhao, *SpecC: Specification Language and Methodology*, Kluwer Academic, 2000.

[12] Synopsys Inc., http://www.synopsys.com

[13] *IEEE Std. 1666-2005, Standard SystemC Language Reference Manual*, IEEE, 2006.

[14] T. Givargis, F. Vahid, and J. Henkel, "System-Level Exploration for Pareto-Optimal Configurations in parameterized System-on-a-Chip," *IEEE Trans. VLSI Systems*, vol. 10, no. 4, 2002, pp. 416-422.

[15] C. Talarico, J.W. Rozenblit, V. Malhotra, A. Stritter, "A new framework for power estimation of embedded systems," *IEEE Computer*, vol. 38, no. 2, 2005, pp. 71-78.

Exploring Clause Symmetry in a Distributed Bounded Model Checking Algorithm

H. Barros - S. Campos
Department of Computer Science
Federal University of Minas Gerais
Av. Antonio Carlos 6627 - ICEx - Pampulha
CEP 31270-010 - Belo Horizonte - Minas Gerais - Brasil
{hbarros, scampos}@dcc.ufmg.br

M. Song - L. Zarate
Department of Computer Science
Catholic University of Minas Gerais
Anel Rodoviario Km 23,5 - Rua Walter Ianni 255
CEP 31980-110 - Belo Horizonte - Minas Gerais - Brasil
{song, zarate}@pucminas.br

Abstract

In recent years new and efficient symbolic model checking algorithms have been developed. One technique, bounded model checking or BMC, has been particularly promising. BMC models the system being verified as a boolean formula whose satisfying assignments provide counterexamples for properties verified. BMC unrolls the system in its multiple iterations. Because of this the structure of the formula representing the system is very symmetric, since all iterations are similar in structure. This work explores this symmetry in a distributed algorithm by postponing the unrolling of the formulas until they are used. This minimizes communication among processors since the formulas transmitted are shorter. Moreover, avoiding the unrolling of conflict clauses *has a more pronounced effect, because due to the symmetric nature of the formula, a conflict clause for one instant in the execution can be applied to* multiple *time instants. As a consequence, short conflict clauses can be unrolled into much more effective clauses, cutting back on the search space significantly. In our experiments we have obtained gains of up to* three *orders of magnitude in verification time and up to* two *orders of magnitude in memory usage in large examples.*

1 Introduction

Symbolic Model Checking (SMC) [1] is an efficient technique for verifying reactive systems. It has proven to be able to find subtle errors in real commercial designs and is gaining acceptance in industrial environments.

Bounded Model Checking (BMC) [2, 3] is a symbolic model checking technique that is based on SAT procedures [4, 5]. In BMC a propositional formula is created in order to represent the model and the properties that will be verified. This formula is then presented to a SAT solver: if it is satisfiable then the model is correct, otherwise it is incorrect — boolean values are used as counter examples to exhibit errors in the system. BMC, based on SAT, can be very efficient in detecting faults in real industrial application models. But, as other model checking algorithms, it suffers from the state explosion problem. Frequently verification cannot be completed because the formula being analyzed is simply too large.

In this paper we propose a way of dealing with this problem. We have implemented a distributed bounded model checking algorithm that reduces time and memory consumption during verification by distributing the work to multiple computers. Moreover, our algorithm also explores the way that BMC constructs the formula that will be sent to the SAT solver in order to take advantage of the inherent symmetry in this formula.

This is accomplished by noticing that a formula constructed by a BMC algorithm is almost completely symmet-

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

ric, since most of the formula consists of repeated applications of the transition relation of the system. In our method, the unrolling of the transition relation that is necessary for BMC is postponed and performed locally by each processor, so communication is minimized. Moreover we apply the same technique to conflict clauses. Since they are often created by the symmetric parts of the formula, the conflict they represent can be "unrolled" to other time instants of system behavior. As a consequence, once a conflicting clause has been found it can be reused not only by different processors, but also by different *time frames*. In our experiments, this method has been quite effective.

We have implemented this method in NuSMV and applied it to several SMV examples. Our experiments have shown gains of up to three orders of magnitude in verification time and up to two orders of magnitude in memory usage. Other examples have shown more modest gains of up to 33 times in execution time and up to 25 times in memory usage, with one particular example with gains of 3.4 and 1.5 in time and memory. Future work includes a better characterization of the types of systems that are better suited for this technique.

This paper is organized as follows. Section 2 presents the related works. Section 4 briefly presents the Bounded Model Checking algorithm, while section5 explains our distributed approach to BMC. Section 6 describes the experimental results and section 7 concludes the paper.

2 Related Work

The use of parallel or distributed algorithms for Bounded Model Checking has been little explored. Although distributed SAT solvers have already been proposed, they usually use concurrent processes to explore disjoint partitions of the search space [6, 7] or they distribute clauses between processes in order to obtain scalability [8, 9]. None of them explore the structure of the clauses.

For example, in [9] the authors implement a algorithm where processes explore sub-formulas of the original SAT formula. However they did not minimize the number of shared variables between partitions. In their implementation whenever a process finishes a SAT deduce step it broadcasts the variables implications to all the other processes. This resulted in communication overhead, up to 90% percent of all messages exchanged between the processes are broadcast.

In [10] a new approach is presented. In this paper the model is split into different *time frames*, where each processor receives a formula that corresponds to a specific value of K, that is, each processors formula determines one possible state in the counterexample sequence. Each processor can then be used to give a partial or local solution to each sub-problem. Once each local solution is found, e.g. for step 3 in the search, a global solution can be derived from local solutions for steps 1 and 2 computed by other processors or an unsatisfiability is detected. This approach has the disadvantage that a possible solution for step 3 can be unreachable, but because partitions are explored independently, it may be computed unnecessarily, and will only be discarded after solutions for steps 1 and 2 are found. In addition to this, all processes must be synchronized, and many decisions such as variable assignment order are centralized, and the algorithm is not as efficient as it could be.

A possible solution to this problem is proposed in [11]. In it a chronological order in the search is used, that is, the algorithm starts determining local solutions from earlier steps before later steps. So, naturally, when step 3 is being processed, the restrictions imposed by steps 1 and 2 have already been processed. Our work follows a similar reasoning up to a certain step, we later break the chronological order in order to increase efficiency as explained later. Different from [11], however, we distribute the workload to speed up verification.

Even though there has been a lot of work on SAT based techniques, some of which as listed below, most of this work does not directly relate to the proposed method. By being orthogonal to the work described here, these techniques can probably be combined with our work to generate even more efficient model checkers.

Currently most of the applications use SAT solvers as black-boxes and no interaction is possible between the applications and the SAT solvers. Application specific knowledge can help a lot in the solution process as demonstrated in [12]. For a particular application, custom implementation of a SAT solver may also be helpful ([13]). In [14] is proposed a SAT based verification technique for threaded C programs. [15] proposes abstraction refinement for BMC. Their technique unifies Proof-based and counterexample-guided abstraction refinements into an abstraction-refinement framework that balances model checking and refinement efforts.

3 Model Checking

Model checking [1] is a formal verification approach by which a desired behavioral system property can be verified over a model through exhaustive enumeration of all states reachable by the application. The model is a labeled state-transition graph. The states correspond to the values of the variables in the program, while the transitions correspond to the passage of time. The model checking process consists of scanning all states in the model to check if the model conforms to the properties.

Formally, the system is represented as a *state-transition graph* $\mathcal{M}$ - a 4-tuple (S, I, A, δ), where S is a set of states, $I \subseteq S$, is a non-empty subset of initial states, A is a set of

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

actions, and $\delta \subseteq S \times A \times S$ is a total transition relation. A run of $\mathcal{M}$ is an infinite sequence $\rho = s_0, s_1 ...$ of states such that $s_0 \in I$ and for all $i \in N, (s_i, A_i, s_{i+1}) \in \delta$ holds for some $A_i \in A$.

Properties are conveniently expressed in temporal logic [16]. Temporal logic is a formalism very useful to describe sequences of transitions between states. One can use temporal logic to reason about the system in terms of occurrences of events.

There are several propositional temporal logic [17]. These logics vary according to the temporal structure (linear or branching time) and the time characteristic (continuous or discrete). Temporal linear logics reason about the time as a chain of time instances. Branching-time logics reason about the time as having many possible futures at a given instance of time.

Time can be continuous or discrete. Time is continuous if between two instances of time there is always another one, otherwise it is classified as discrete. In this work is used a branching-time and discrete logic known as Computation Tree Logic (CTL [1]). CTL is defined on state transition graphs. The graph structure is unwound into an infinite tree rooted at the initial state. Paths in this tree represent all possible computations of the system being modeled.

CTL provides operators to be applied over the paths formed by the computation tree. When these operators are specified in a formula they must appear in pairs and in a specific order: *path quantifier* followed by *temporal operator*. A path quantifier defines the scope of the paths over which a formula f must hold. There are two path quantifiers: **A**, meaning **all** paths; and **E**, meaning **some** path. A temporal operator defines the appropriate temporal behavior that is supposed to happen along a path relating a formula f. The temporal operators are the following: **F** ("in the future" or "eventually") - starting from the root, f holds in some state of the path; **G** ("globally" or "always") - starting from the root, f holds in all states of the path; **U** ("until") - there is a state s in the path where a formula g is satisfied and all predecessor states of s satisfies f. **X** ("next time") - starting from the root, f holds in the second state of the path.

If f and g are CTL formulas, then $\neg f$, $f \vee g$, $f \wedge g$, AFf, EFf, AGf, EGf, A[fRg], E[fRg], A[fUg], E[fUg], AXf, EXf are CTL formulas. Some examples of CTL formulas are given to illustrate the expressiveness of the logic: $AG(req \rightarrow AF\ ack)$ - it is always the case that if the signal req is high, then eventually ack will also be high; $EF(started \wedge \neg ready)$ - it is possible to get to a state where $started$ holds but $ready$ does not hold.

4 Bounded Model Checking

In *Bounded Model Checking*, or BMC, a propositional formula is created to check if the model has a finite sequence of states that satisfies some property[2, 3]. A limit (K) is imposed on the size of the sequence (number of states to be reached). This limit can be interpreted as the maximum size of the counter-example.

According to BMC any model can be translated according to the following equation:

$$[M, P]_k := I(s_0) \wedge \left(\bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \right) \wedge [P]_k^0 \quad (1)$$

where

- M is a *Kripke* structure;
- P is a temporal logic formula representing the property being verified;
- k is the depth of the search algorithm;
- $I(s_0)$ is a formula representing the set of initial states.
- $T(s_i, s_{i+1})$ is the formula representing the transition relation of the system where s_i is the current state of the transition and s_{i+1} is the next state.
- $[P]_k^0$ is a property to be verified.

The propositional formula $[M, P]_k$ is satisfiable if, and only if, P is valid in some path of M. SAT solvers (see code) are used to determine the satisfiability of this formula.

```
 Input: d - decision level
 Output:
  SAT(): {SATISFAZIVEL,INSATISFAZIVEL}
  Decide(): {ATRIBUI, TODAS-ATRIBUIDAS}
  Deduz(): {OK, CONFLITO}
  Diagnostica(): {INVERTE, RETROCEDE}
  Global variable: B - backtracking

procedure SAT(d)
  if (Decide(d) == TODAS-ATRIBUIDAS) then
  return SATISFAZIVEL;
  end if
  while (TRUE) do
   if (Deduz(d) != CONFLITO) then
     if (SAT(d + 1) == SATISFAZIVEL) then
        return SATISFAZIVEL;
     else if (B < d || d == 0) then
         Apaga(d);
         return(INSATISFAZIVEL);
     end if
   end if
   if (Diagnostica(d) == RETROCEDER) then
     return INSATISFAZIVEL;
   end if
  end while
end procedure
```

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

5 A Distributed Approach to BMC

This section describes the proposed approach to distributed BMC. As in [10], we partition the model into different *time frames*, where each time frame corresponds to one cycle of the system under verification. An initial partition is generated using a partial assignment for the initial state (this partition also takes into consideration the property being verified, as will be seen later). The processor that generates this assignment is called *primary solver*. This partition is sent to a *secondary solver*, which continues generating partial assignments for the subsequent time frames up to the kth time frame. The primary solver creates and starts several secondary solvers, each with a different partial assignment for time frame 1. Each secondary solver then determines a complete satisfying assignment for the formula. If a secondary solver determines that the formula is unsatisfiable (for the given partial assignment for time frame 1), it sends the conflict information to the primary solver that informs all other solvers about the conflict. Figure1 depicts our approach.

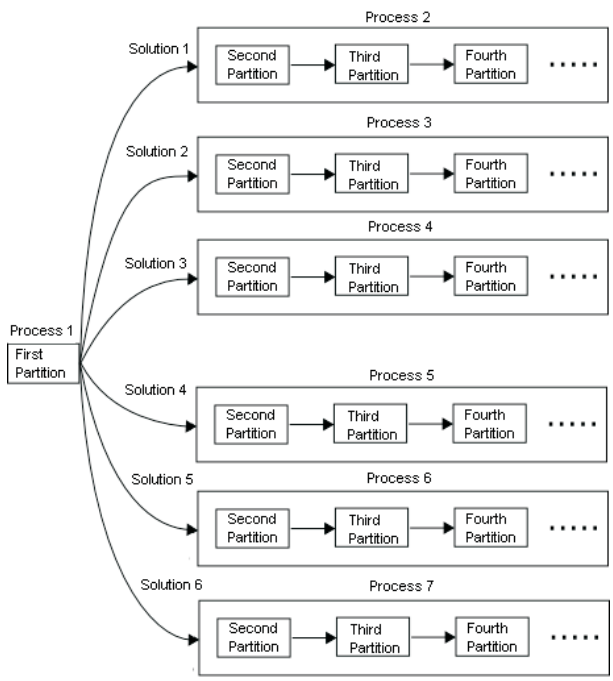


Figure 1. Partitions generated from BMC clauses.

As noted the proposed solution splits the problem in such a way that the same set of clauses can be used for different partitions. This guarantees that less memory is used to represent the problem. Also, as each process explores partition two to partition K by itself, less communication is needed.

5.1 Clause Partitioning and Partition Symmetry

Clauses are partitioned with the objective to minimize the number of shared variables between any two partitions, and to better take advantage of the symmetrical nature of a BMC formula. Remember that the following formula is used to represent the model:

$$[M,P]_k := I(s_0) \wedge [P]_k^0 \wedge \left(\bigwedge_{i=0}^{k-1} T(s_i, s_{i+1}) \right) \quad (2)$$

Note that the sub-formula that represents the initial state and the property is not symmetric to other sub-formulas. However, all other sub-formulas are symmetric, and in fact each $T(s_i, s_{i+1})$ sub-formula is the same formula modulo variable renaming. Based on this observation, we partition the formulas such that the first partition contains $I(s_0) \wedge [P]_k^0$, and for all $i > 1$, partition i contains $T(s_i, s_{i+1})$. For example, expanding the formula above for k = 6, we generate:

$$\begin{aligned}
[M,P]_6 &:= I(s_0) \wedge [P]_6^0 \wedge \left(\textstyle\bigwedge_{i=0}^{6-1} T(s_i, s_{i+1}) \right) \\
[M,P]_6 &:= I(s_0) \wedge [P]_6^0 \wedge T(s_0, s_1) \wedge T(s_1, s_2) \wedge \ldots \\
&\quad \wedge T(s_3, s_4) \wedge T(s_4, s_5) \wedge T(s_5, s_6) \\
[M,P]_6 &:= \underbrace{I(s_0) \wedge [P]_6^0}_{P1} \wedge \underbrace{T(s_0, s_1)}_{P2} \wedge \underbrace{T(s_1, s_2)}_{P3} \wedge \underbrace{\ldots}_{P4} \\
&\quad \wedge \underbrace{T(s_3, s_4)}_{P5} \wedge \underbrace{T(s_4, s_5)}_{P6} \wedge \underbrace{T(s_5, s_6)}_{P7}
\end{aligned}$$

$I(s_0)$ is the first partition, $T(s_0, s_1)$ the second partition, $T(s_1, s_2)$ the third partition and so on. Note that each partition just shares state variables where a cut occurs. In the example a cut was done in state s_1 and these variables are the only ones shared between the second and third partition.

Note also that the formula has been divided into seven partitions $P1$ to $P7$. The first one is composed of $I(s_0) \wedge [P]_k^0$ and it is denoted as the *primary partition*. The others are referred to as *secondary partitions*. The symmetry between the secondary partitions is clear, the only difference between formula $T(s_1, s_2)$ and $T(s_2, s_3)$ are the indexes. For example:

$$\begin{aligned}
T(s_i, s_{i+1}) : &\quad (a_{i+1} \leftrightarrow \neg a_i) \wedge (b_{i+1} \leftrightarrow a_i \bigoplus b_i) \\
T(s_1, s_2) : &\quad (a_2 \leftrightarrow \neg a_1) \wedge (b_2 \leftrightarrow a_1 \bigoplus b_1) \\
T(s_2, s_3) : &\quad (a_3 \leftrightarrow \neg a_2) \wedge (b_3 \leftrightarrow a_2 \bigoplus b_2)
\end{aligned}$$

The similarity between partitions has many advantages, one of the most important ones being that it allows for sharing and replication of conflict clauses between partitions in a natural and efficient way. For example, a conflict clause

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

generated from formula $T(s_1, s_2)$ can be used in the solution of $T(s_2, s_3)$ by just changing indexes. Once a secondary partition has found a conflict clause it can be shared not only between solvers, but also between *time frames*. For example, suppose that in a secondary partition a conflict clause was found: $\pi = (\neg x_4 \vee y_7 \vee z_5)$. One can certainly state, based on the partition symmetry, that the clause $\pi = (\neg x_3 \vee y_6 \vee z_4)$ will be also a conflict clause (we name such a clause as replicated clause). Our experiments show that these conflict clauses are usually small and are therefore found and communicated to other solvers very fast. All conflict clauses are sent to the primary solver and inserted into the first partition reducing drastically the size of the search tree.

5.2 The Distributed Model Checker

In this section, the distributed model checker is presented. The main idea is to use secondary solvers to explore clause symmetry. The primary solver finds a partial solution (primary solution) and the solver manager deals with all operations and communications. Figure 2 shows the main components of the distributed model checker. Each component is briefly described:

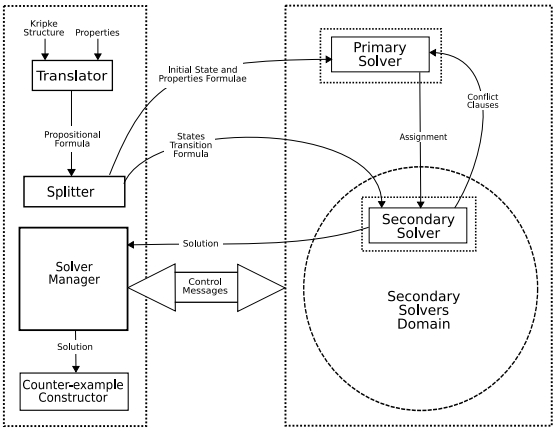


Figure 2. The Distributed Model Checker.

- Translator: converts the kripke structure and properties into a propositional formula;
- Splitter: creates the primary and secondary partition — it instantiates each solver and dispatches to then each correspondent partition.
- Primary Solver: searches for a primary solution. It interacts with the Solver Manager in order to find a free secondary solver which will receive its partial solutions;

```
/* Primary Partition */
be_ptr primary_beProb;
/* Initiating Primary Partition Problem */
primary_beProb = Bmc_Model_GetInit0(be_fsm);
bmc_add_be_into_solver_positively(
SAT_SOLVER(primary_zchaff),
SatSolver_get_permanent_group(
   SAT_SOLVER(primary_zchaff)),
   primary_beProb,
   vars_mgr);
...
/* Start problems generations: */
for (increasingK = k_min;
     (increasingK <= k_max && !found_solution);
      ++increasingK) {
  SatSolverGroup additionalGroup =
    SatIncSolver_create_group(
       SAT_INC_SOLVER(primary_zchaff));
  /* Primary Partition problem in BE format */
    be_ptr properties_beProb;
  ...
  /* Adding Properties Formulae
     to Primary Partition Problem */
  /* Add LTL tableau to an additional
     group of the secondary solver */
  properties_beProb = Bmc_Tableau_GetLtlTableau(
       be_fsm, bltlspec,increasingK, l);
  bmc_add_be_into_solver_positively(
     SAT_SOLVER(primary_zchaff),
       additionalGroup,
       properties_beProb,
       vars_mgr);
  ...
  SatIncSolver_destroy_group(
     SAT_INC_SOLVER(primary_zchaff),
          ddditionalGroup);
  ...
}
```

- Secondary Solvers: search for a solution based on the partial solution found by the primary solver. For example: one can obtain the solution to $T(s_0, s_1)$ based on the primary solution. If a solution is found then try it for $T(s_1, s_2)$. If for any transition $T(s_{i+1}, s_{i+2})$ no solution is found a new search is done for $T(s_i, s_{i+1})$. If $T(s_0, s_1)$ has no solution then the problem is unsatisfiable. If a solution to $T(s_{k-1}, s_k)$ is found then the problem is satisfiable.

  It is important to notice that secondary solvers run asynchronously, they do not interact with each other.

```
/* Secondary Partition */
be_ptr secondary_beProb;
/* Initiating Secondary Partition Problem */
/* Unroll the transition relation to
   the fixed frame 1 */
secondary_beProb =
    Bmc_Model_GetUnrolling(be_fsm,0,1);
bmc_add_be_into_solver_positively(
```

```
SAT_SOLVER(secondary_zchaff),
SatSolver_get_permanent_group(
      SAT_SOLVER(secondary_zchaff)),
secondary_beProb,
vars_mgr);
```

- Solver Manager: coordinates all the operations between primary and secondary solvers.
- Counter-Example Constructor: translates a solution from a propositional formula to a path in the model;

6 Experimental Results and Conclusions

This section presents a comparative study of the distributed approach proposed with sequential BMC methods based on SAT. In this work we have used NuSMV [18] with the ZChaff SAT solver [19] both for the sequential and the distributed algorithms.

In order to distribute tasks among processes the LAM/MPI paradigm [20, 21] has been adopted. MPI, or *Message Passing Interface*, is a standard for communication among nodes running a parallel program on a distributed memory system. MPI is a library of routines that can be called from C programs. MPI is both portable (it has been implemented for almost every distributed memory architecture) and fast (usually each implementation is optimized for the hardware it runs on).

All tests have been performed on a cluster composed of 22 nodes, even though at all examples only 5 secondary solvers have been used. Each node is a Intel Pentium IV processor, 3.00 GHz, 1 GB RAM, 80.0 GB disk (ATA) and Fast Ethernet network controller (100 Mbps). All systems were running under the Linux operating system. The examples can be found in the NuSMV distribution [18]:

- periodic.smv - a data driven pipeline example;
- p-queue.smv - a priority queue example;
- gigamaxltl.smv - a model of the GIGAMAX cache coherence protocol;
- dme5.smv, dme8.smv, dme10.smv, dme20.smv - a asynchronous version of a distributed mutual exclusion algorithm with 5, 8, 10 and 20 cells.

The tables below show the NuSMV models and the corresponding number of boolean variables. For each NuSMV model the sequential and distributed execution have been monitored until response time became impractical. The following information have been collected (for the same values of k for each model): execution time, number of SAT clauses generated and memory used.

Model	Boolean Variables	Max K	Exec. Time (s)	Mem (MBytes)	Clauses (x1000)
periodic.smv	35	95	21185	13.5	227.57
p-queue.smv	43	19	19387	51.9	896.64
gigamax_ltl.smv	49	61	16043	268.8	312.86
dme5.smv	180	49	3721	397.9	319.55
dme8.smv	287	32	8463	280.8	668.19
dme10.smv	360	30	9605	428.3	1051.5
dme20.smv	720	20	7312	432.5	1394.12

Table 1. Sequential Algorithm Results

Model	Boolean Variables	Max K	Exec. Time (s)	Primary Solver		Secondary Solvers (Max)	
				Mem (MBytes)	Clauses (x1000)	Mem (MBytes)	Clauses (x1000)
periodic.smv	35	95	69	5.1	73.01	2.9	29.84
p-queue.smv	43	19	15	< 1	5.9	< 1	46.89
gigamax_ltl.smv	49	61	4	< 1	4.0	< 1	3.7
dme5.smv	180	49	1103	263.7	225.96	1.4	8.1
dme8.smv	287	32	256	15.2	131.59	1.7	8.0
dme10.smv	360	30	419	23.7	157.97	2	9.29
dme20.smv	720	20	435	16.8	180.99	2.3	13.48

Table 2. Distributed Algorithm Results

Table 3 summarizes the gains obtained by our method. The **Time**, **Mem** and **Clauses** columns are the sequential values divided by the distributed values of the previous tables. For the Mem and Clauses columns we have chosen the maximum of primary and secondary solver values. These gains have been calculated from values of actual experiments, while the previous tables show rounded values to increase readability.

Model	Boolean Variables	Max K	Exec. Time	Mem	Clauses
periodic.smv	35	95	307	2.62	3.1
p-queue.smv	43	19	1292	115	19.1
gigamax_ltl.smv	49	61	4010	621	78
dme5.smv	180	49	3.4	1.5	1.4
dme8.smv	287	32	33	18	5
dme10.smv	360	30	23	18	6.65
dme20.smv	720	20	16.8	25.7	7.7

Table 3. Gains of the Proposed Method

We can see from the tables above that the proposed method does indeed improve significantly on the results of the sequential method. We have obtained gains of up to *three orders of magnitude* in verification time and up to two orders of magnitude in memory usage. The first three examples show that the method proposed can give substantial gains for different types of systems, even in cases with deep searches. The `dme` examples, however, show also that for some examples the depth of the search can affect the gains, with `dme5` having the deepest search and the worst results. The depth of the search is probably the reason for this, since all `dme` examples are very similar.

Figures 3, 4, and 5 show resource usage for the *periodic.smv* example during its execution. These are typical figures for the examples verified. Figures for all other examples can be found in [22]. It is interesting to see that

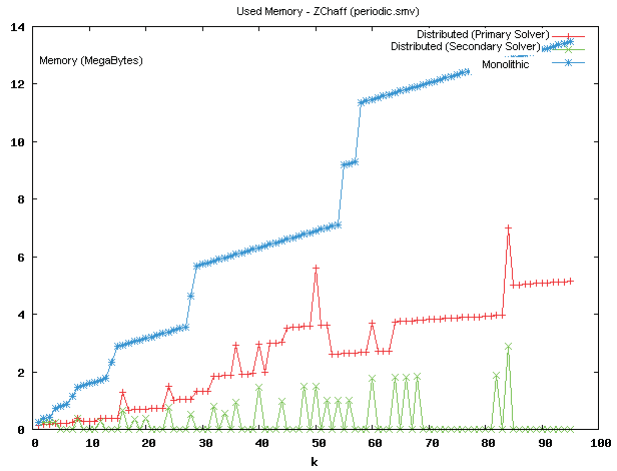


Figure 3. Experimental Result

the primary solver uses more resources that the secondary solvers all the time (this is even more pronounced in other examples). This would indicate that most of the work is being done by the primary solver. But clearly this is inconsistent with the gains obtained. An explanation is that the secondary solvers are able to determine short conflict clauses very fast, transmitting them back to the primary solver. But these clauses are not only sent to multiple secondary solvers, but are also "unrolled" in multiple time frames, cutting back significantly on the amount of the work done by other secondary solvers.

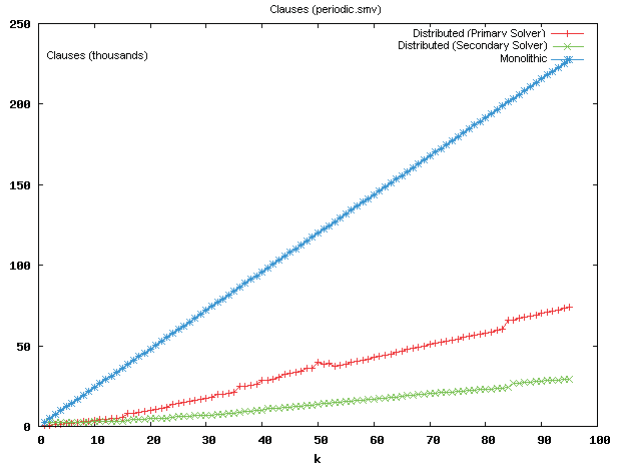


Figure 4. Experimental Result

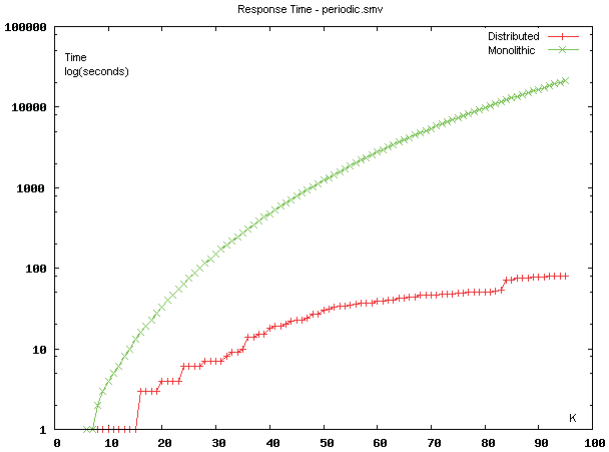


Figure 5. Experimental Result

7 Conclusions

Bounded model checking is an efficient technique for verifying reactive systems. However, as all model checking techniques, it suffers from state explosion, and frequently cannot handle real systems. In this paper we propose a new method for verifying reactive systems that improves on BMC by distributing the work through a cluster of computers. Our method also explores the symmetry of the clauses generated by the BMC algorithms to increase efficiency. We consider that most of the clauses of a BMC problem are symmetric with respect to time since the same transition relation is applied at all time instants. By exploring this characteristic, we are able to generate conflict clauses that are short (and therefore found fast), but that can be applied not only to multiple processors but also to multiple time frames.

Our experiments have shown that the proposed method can achieve gains of up to three orders of magnitude in some examples in verification time and up to two orders of magnitude in memory usage. Future work includes a better characterization of the types of systems that are better suited for this technique, and the study of other ways of exploring the symmetry of the clauses in verification. We also plan on studying in more detail how much of the gains come from the distributed algorithm, and how much come from the use of symmetry.

References

[1] E. M. Clarke and O. Grumberg and D. A. Peled. *Model Checking*. The MIT Press, Cambridge, Massachusetts, 1999.

[2] A. Biere and A. Cimatti and E. Clarke and Y. Zhu. Symbolic Model Checking without BDDs. *Lecture Notes in Computer Science*, 1579:193–207, 1999.

[3] A. Biere and A. Cimatti and E. M. Clarke and M. Fujita and Y. Zhu. Symbolic Model Checking using SAT procedures instead of BDDs. In *Proceedings of Design Automation Conference (DAC'99)*, 1999.

[4] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms.* MIT Press, 2001.

[5] I. Gent and T. Walsh. The search for Satisfaction. Technical report, 1999.

[6] H. Zhang and M. P. Bonacina and J. Hsiang. PSATO: a Distributed Propositional Prover and its Application to Quasigroup Problems. *Journal of Symbolic Computation*, 21(4):543–560, 1996.

[7] Y. Feldman and N. Dershowitz and Ziyad Hanna. Parallel Multithreaded Satisfiability Solver: Design and Implementation. In *Proceedings of the 3rd International Workshop on Parallel and Distributed Methods in Verification*, 2004.

[8] M. Bhm and E. Speckenmeyer. A Fast Parallel SAT-solver - Efficient Workload Balancing. In *Third International Symposium on Artificial Intelligence and Mathematics*, 1994.

[9] Y. Zhao and S. Malik and M. W. Moskewicz and C. F. Madigan. Accelerating boolean satisfiability through application specific processing. In *ISSS*, pages 244–249, 2001.

[10] M. Ganai and A. Gupta and Z. Yang and P. Ashar. Efficient Distributed SAT and SAT-Based Distributed Bounded Model Checking. *Lecture Notes in Computer Science*, 2860:334–347, 2003.

[11] O. Strichman. Tuning SAT Checkers for Bounded Model Checking. In *Computer Aided Verification*, pages 480–494, 2000.

[12] O. Strichman. Pruning techniques for the SAT-based Bounded Model Checking Problem. *11th Advanced Research Working Conference on COrrect Hardware Design and Verification Methods (CHARM'01)*, 2001.

[13] M. Ganai and L. Zhang and S. Malik. COmbining Strengths of Circuit-based and CNF-based Algorithms for a High-Performance SAT Solver. *Design Automation Conference (DAC'02)*, ():, 2002.

[14] Ishai Rabinovitz and Orna Grumberg. Bounded model checking of concurrent programs. *CAV*, pages 82–97, 2005.

[15] Anubhav Gupta and Ofer Strichman. Abstraction refinement for bounded model checking. *CAV*, pages 112–124, 2005.

[16] Michael R.R. Huth and Mark D. Ryan. *Logic in Computer Science - Modelling and reasoning about systems*. Cambridge University Press, 2000.

[17] P. Bellini, R. Mattolini, and P. Nesi. Acm computing surveys 32. In *Temporal logic for real-time system specification*, 2000.

[18] A. Cimatti and E. Clarke and E. Giunchiglia and F. Giunchiglia and M. Pistore and M. Roveri and R. Sebastiani and A. Tacchella. NuSMV 2: An Open-Source Tool for Symbolic Model Checking. In *In International Conference on Computer-Aided Verification*, 2002.

[19] SAT Research Group, Princeton University. zChaff. `http://ee.princeton.edu/~chaff/zchaff.php`.

[20] R. Hempel. The MPI Standard for Message Passing. *Lecture Notes in Computer Science*, 797:247–252, 1994.

[21] G. Burns and R. Daoud and J. Vaigl. LAM: An Open Cluster Environment for MPI. In *Proceedings of Supercomputing Symposium*, pages 379–386, 1994.

[22] H. Barros. A distributed approach to bounded model checking. Master's thesis, Universidade Federal de Minas Gerais, 2005.

Reasoning about cryptographic protocols in observational theories

Imen Zaabar
Tunisia Polytechnic School(EPT)
Marsa, Tunisia
zaabarim@egr.msu.edu

Narjes Berregeb
National Institute of Applied
Sciences and Technology(INSAT)
Tunis, Tunisia
narjes.benrajeb@topnet.tn

Abstract

A lot of models have been applied to the analysis of cryptographic protocols. Some of them formalize security properties as reachability and some others express them as observational equivalences. In this paper, our intent is to present the main ideas for reasoning about cryptographic protocols in the framework of observational theories. We model protocols by Term Rewriting systems and express secrecy and authentication properties by observational equivalences, in a way close to spi-calculus [3].

Keywords— Security Protocols, Verification, Term rewriting, Observational equivalence.

1. Introduction

Cryptographic protocols aim at achieving goals like authentication and secrecy. They become more and more important with the development of communications via public networks. Unfortunately, many of them are subjects to very subtle attacks that cannot be detected by informal reasoning. Consequently, there has been a growing interest in applying formal methods for verifying cryptographic protocols.

A lot of models have been applied to the analysis of some particular cryptographic protocols. Some of them formalize security properties as reachability [4, 19] while others express them as observational equivalence like in Spi-calculus [3]. Reachability properties can be stated as follows: we say that a protocol does not reveal its secret, if there is not an execution trace where the intruder can deduct the secret. As for observational equivalence, it can be stated as follows: we say that a data s is secret if the session that contains the data s is *indistinguishable* of all session containing a data s' instead of s. In this context, Spi-calculus is interesting because it allows to reason on different levels of a system abstraction. The intruder is modelled as an environment which is a non-deterministic process. Unfortunately, proving properties in spi-calculus is hard; currently there is no tool allowing to automate these proofs.

Our aim is to use ideas close to the ones used in Spi-calculus [3] to express secrecy and authentication properties in the framework of observational algebraic specifications. Our approach is amenable to automation, especially if using former ideas for automating observational proof [9, 8]. Observational equivalence can be seen as a relaxing of classical equivalence: two objects are considered indistinguishable unless they are proved distinguishable under some experiments called observable context.

In our model, the intruder is similar to Dolev-Yao model [13]. In fact, the intruder, which is not explicitly modelled, can accede and manipulate all and only the public (or observable) data. Its actions are represented by observable contexts that can be applied on any message transmitted through public network. Proving that a message **M** is maintained secret during the running of the protocol comes to prove that an intruder cannot distinguish this message M from another message **M'** during all the running steps.

The paper is structured as follows: Section 2 presents basic notions of observational algebraic specifications. Section 3 gives an overview of related works. In section 4, we present our modelling of cryptographic protocols. Section 5 shows how to formalize security properties in the framework of observational algebras. In section 6, we give some examples of protocol analysis. Concluding remarks are presented in Section 7.

2. Basic notions

We assume that the reader is familiar with basic notions of cryptographic and security protocols (public and symmetric key cryptography) [18], of rewriting [8]

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

and of algebraic specifications [14].
Since algebraic concepts and tools are more and more used in Computer science, observational algebraic specifications are an appropriate formalism for the specification of the visible behavior of systems in states as well as the visible properties of abstract data. This formalism compounds with the paradigm *object* used in *oriented object languages* [17]. Moreover, observational theories [6] has a major advantage since they allow reasoning on several abstraction levels of a system (program, protocol, etc...). So, they allow to make abstractions of the internal details of the system to implement and focalize only on the external behavior of this *observable* system. Therefore, observational concepts are fundamental in formal methods, i.e. system verification in general, and cryptographic protocols in particular.

There are various notions of observability involving observation techniques based on sorts, operations, terms or formulae [7]. To rely on some observational technique means to choose which kind of objects we observe and how we observe them. In our approach, we choose to use observation techniques based on sorts and operators. In this case, an observational signature is a generalization of a standard algebraic signature with a distinguished set of observable sorts and observer operations, formally denoted by $\Sigma_{obs} = (\Sigma,\ S_{obs},\ OP_{obs})$ where Σ is a signature, S_{Obs} is a set of observable sorts and OP_{Obs} is a set of observers [15, 19]. An observational signature determines a set of observable contexts (definition 2.2), which represent those experiments which allow us to distinguish elements by the given observers [15, 19].

Definition 2.1 (Context) *Let T(F,X) be a term algebra and $\Sigma = (S, F)$ its signature.*

1. *a context over Σ (or Σ-context) is a non-ground term $c \in T(F, X)$ with a distinguished linear variable called the context variable of c. To indicate the context variable occurring in c, we often write $c[z_s]$ instead of c, where s is the sort of z_s. A variable z_s of sort s is a context called empty context of sort s. We can also simply denote a context $c[z_s]$ by $c[\_]$*

2. *the application of a context $c[z_s]$ to a term $t \in T(F, X)$ of sort s is denoted by $c[t]$ and is defined as the result of the replacement of z_s by t in $c[z_s]$. The context c is said to be applicable to t.*

Definition 2.2 (Observable context) *An observable context is a context whose sort belongs to S_{obs} and all its operators belongs to OP_{obs}.*

We denote by C_{obs} the set of observable context.

Elements which cannot be distinguish by the experiments of an observational signature are considered to be observationally equal (definition 2.3).

Definition 2.3 (Observational equivalence) *Let t_1 and t_2 two term in $T(F, X)$, then we say that t_1 and t_2 are observationally equal (or equivalent)and denoted by $\approx_{obs}$ if and only if for all observable context c we have $c[t_1] = c[t_2]$. Formally, we write*

$$t_1 \approx_{obs} t_2 \quad iff \quad \forall\ c\ \in\ C_{obs} \quad c[t_1] = c[t_2].$$

Example Let Σ_{obs} be an observational signature for bank accounts with observer *bal* and an operation *update* [15]: **sorts**$\{account,\ int\}$, **observable sorts**$\{int\}$, **observers**$\{bal{:}account{\rightarrow}int\}$
operations$\{new{:}{\rightarrow}account,\ update{:}account{,}int{\rightarrow}account\}$
Observable contexts are $z_{account}$, $bal(z_{account})$.

3. Related works

In this section, we will discuss the objectives and advantages of Spi-calculus. Details of the syntax and language are given in [2]. The Spi-calculus is a language of processus applied to cryptographic protocols. It's like the CSP language (Communicating Sequential Processes) [19] and the language developed by R.Amadio et al. in [5]. He allows, in particular, to express the mail and the reception of messages, the creation of nonce and the replication of processes.
The languages of processes introduce a major difference with other models of cryptographic protocols. These models express the security properties as reachability properties: there is a valid trace such that the intruder can deduct secret or the process can attain the state *error*. For the languages of processes such as the Spi-calculus, the secret as well as the other properties can be expressed in form of observational equivalences and the intruder can be represented by any random process in the syntax. we say that a protocol $P(s)$ is secret if $P(s)$ and $P(s')$ are indistinguishable whatever the process in parallel. Such property takes into account the capacity of an intruder, for example, to compare messages, which is not the case in other models.
Unfortunately, it is very difficult to show the observational equivalence of two processes. Therefore, there is no tool allowing to automate the proof of observational equivalence between processus.
M.Boreale et al. [11] shows in a restrained context that observational equivalence is similar to a form of bisimulation using a static equivalence. However, the considered framework does not allow to express the

composite keys nor the asymmetrical encoding. On the other hand, M.Abadi and C.Fournet [1] established a similar result in an extremely general context. But even static equivalence is infinite in this framework.

4. Protocol modelling

4.1. Global state

Our modelling of cryptographic protocols is close to the one in [16]. The protocol is modelled by a set of transition rules applied on a multiset of objects representing a global state. The global state contains both sent messages and expected ones. A send operator is denoted by $send(i, s', s, r, t)$ where i is the protocol step, s' is the real sender, s is the official sender, r is the receiver and t is the message. An expected message is denoted by $wait(i, s, r, t)$ with similar meaning for the fields i, s, r and t.

4.2 Rewriting system of the protocol

Rewriting rules:
A transition rule (denoted by $l \rightarrow$ r) can be stated as follows: let B a principal waiting a message from A at step i, and A the one sending a message M to B at step i. The global state at step i can be denoted by $\{send(i, A, A, B, M).wait(i, A, B, x_M)\}$. When A sends the message, he changes his state to waiting a new message at the k_i^{th} step,where:

$k_i = \min\{j|\ j > i\ et\ R_j = R_i\}$ if not empty
$\quad = \min\{j|\ j \leq i\ et\ R_j = R_i\}$ otherwise.

with R_i is the receiver at the ith step. When receiving message M from A, B checks whether he received what he was expecting and then composes and sends a message at the next step. To summarize, a transition rule is as follows: the next message in the protocol is composed and sent, and the next expected one is built too.

Protocol running
The initial state is denoted by E_1. Let $N = \{1, 2, \ldots\}$, the execution of the protocol is traduced by the application of transition rules to a global sate E_i (where $i \in N^+$) to generate a new global state E_j at the next step. We denoted it by $E_i \rightsquigarrow E_j$ where j = ß1.

Message rewriting system
Manipulation operators of messages M_i, where i is the number of step, are defined as follows:

- $K(\_)$: encryption operators with public key,
- $SK(\_)$: encryption operators with symmetric key,
- $< -, - >$: messages composition,
- $\pi_1(\_)$ et π_2: projections operators,

The Message rewriting system is defined according to the following rules:

$$\begin{array}{rcl@{\qquad}rcl} SK(SK(x)) & \rightarrow & x & K^{-1}(K(x)) & \rightarrow & x \\ K(K^{-1}(x)) & \rightarrow & x & \pi_1(< x_1, x_2 >) & \rightarrow & x_1 \\ \pi_2(< x_1, x_2 >) & \rightarrow & x_2 & & & \end{array}$$

Observable signature
Our idea is to divide the specification signature into observable operators that are accessible to any observers, in particular the intruder and non-observable ones that are accessible only by the legitimate actors of the protocol. For instance, some of the observable operators can be: principal names, public key (denoted by K), encryption operators with public key (denoted by K(.)), composition and decomposition operators and send operator. Non-observable operators can be: a private key (denoted by K^{-1}), decryption operators with private key (denoted by $K^{-1}(.)$).

Intruder
A test or an experiment formalizes the idea of one or several actions which an intruder can make on messages circulating in the network. Formally and within the observational framework, a test is an observable context representing the series of action which an intruder can make on an intercepted message. Note that it is possible to use similar techniques in [8] to schematize in a finite way all the observable contexts that the intruder can apply. Thus, a general context has to contain the send operator Send(-,-,-,-,-) as a last subcontext so that there is no blocking in the execution of a protocol. Besides, because an intruder does not follow the protocol steps, the number of step generated by the intruder is x. Therefore, a general context can be stated as follow:

$$send(x, I, x_s, x_r, C[\_])$$

where :

- x: step number;
- I: Intruder name;
- x_s: sender name in the ith step,
- x_r: receiver name in the ith step,

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

- C[-]: intruder action (observable context) on the intercepted message in the ith step.

To summarize, an intruder can modify a global state E_i by applying a context on intercepted messages. We denote in this case the new global state by E_{ij} where $j > 1$.
The set of all the executions of the protocol could be schematized in figure 1.

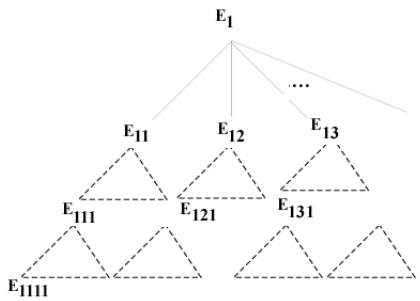


Figure 1. set of all the executions of the protocol

We denote by $E_i(M)$ a state of a protocol execution where M is a secret message.

5. Security properties

Security properties are stated in terms of observational equivalences. In fact, to prove that a secret M is not compromised during the protocol execution, it is necessary to prove that an external observer cannot distinguish between an execution of the protocol with a secret M and another one with M' instead of M. Formally, secrecy is stated as follows:

$$\forall M, M',\ \forall j \in N^+,\ E_j(M) \approx_{obs} E_j(M').$$

To prove such equivalence, one has to apply all possible observable contexts representing actions done by an intruder. to prove the secrecy property, one has to compare messages circulating in the network and as a consequence they are indistinguishable by the intruder if they are encrypted. This hypothesis could be stated by the lemmas 5.1 and 5.2:

Lemma 5.1 *Let K be a secret key. Then*

$$\forall M, M'\ K(M) \approx_{obs} K(M')$$

Proof 5.1 *The lemma results from the hypothesis of perfect encoding. Since the intruder does not have the private key K^{-1}, there is no observable context that differentiate M from M'.*

Lemma 5.2

$$\forall M, M'\ ciphertextes\ if\ M \neq M'\ then\ M \not\approx_{obs} M'$$

Proof 5.2 *Let $z_{obs}[\_]$ be the empty context. Then, $z_{obs}[M] = M \neq M' = z_{obs}[M']$. Therefore, $M \not\approx_{obs} M'$.*

Authenticity property is also expressed as an observational equivalence.It compares the given protocol with another that serves as a specification. For example, let A and B be two principals. The specification can be stated as follows: the principal A is as usual, but the principal B is replaced with a variant B^{spec}; this variant receives a message from A and then acts like B when B receives this message. To prove authenticity comes to prove that, whatever the intruder can do, B^{spec} (the expected sender), is always the real sender. Formally, we can say that:

$$\forall i, {E_i}^{spec} \approx_{obs} E_i.$$

6. Examples

6.1. Example 1: Secrecy attack

Our first example is extremely basic. In this example, there are two principals A and B. The protocol is simply described as follows: A sends a secret message M under the private key K^{-1}, i.e $K^{-1}(M)$ to B. Then, B send a message under the public key K, i.e $K(M)$when he receives the message from A. This is specified as follows:

Message 1: $A \rightarrow B : K^{-1}(M)$
Message 2: $B \rightarrow A : K(M)$

The rewrite rules of the protocol are:

$$\left[\begin{array}{l} wait(0,x_1,A,x_2) \rightsquigarrow \\ send(1,A,A,B,K^{-1}(M)) \cdot wait(0,x_1,A,x_2) \end{array}\right] \quad (R1)$$

$$\left[\begin{array}{l} send(1,A,A,B,K^{-1}(M)) \cdot wait(1,x_3,B,x_4) \rightsquigarrow \\ send(2,B,B,A,K(M)) \cdot wait(1,x_3,B,x_4) \end{array}\right] \quad (R2)$$

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

$$\left[\begin{array}{l} send(2,B,B,B,K(M))\cdot wait(2,B,A,K(x_M))\rightsquigarrow \\ wait(0,x_1,A,x_2) \end{array}\right] \quad (R3)$$

The execution of the protocol is as follows:
$E_1 = wait(0, x_1, A, x_2) \cdot wait(1, x_3, B, x_4)$
$\rightsquigarrow$
$E_{11} = send(1, A, A, B, K^{-1}(M)) \cdot wait(2, B, A, K(x_M)).wait(1, x_3, B, x_4)$
$\rightsquigarrow$
$E_{111} = send(1, B, B, A, K(M)) \cdot wait(1, x_3, B, x_4) \cdot wait(2, B, A, K(x_M))$
$\rightsquigarrow$
$E_{1111} = wait(1, x_1, A, x_2) \cdot wait(1, x_3, B, x_4)$
Let $C_{obs} = send(0, I, A, B, K[\_])$ be the observable context (thus, applied by an intruder) to prove secrecy, then we have also:
$E_{11} \rightsquigarrow$
$E_{112} = send(0, I, A, B, M) \cdot wait(1, x_3, B, x_4)$
$\cdot wait(0, x_1, B, x_2)$
to prove the attack, we have to prove that

$$\exists M, M', E_{112}(M) \not\approx_{obs} E_{112}(M').$$

Let $E(M)$ and $E(M')$ be:

$$\text{E(M)} = send(x, I, A, B, K(K^{-1}(M))) \rightarrow send(x, I, A, B, M)$$

$$\text{E(M')} = send(x, I, A, B, K(K^{-1}(M'))) \rightarrow send(x, I, A, B, M')$$

Since $M \not\approx_{obs} M'$, according to lemma 5.2,

$$E(M) \not\approx_{obs} \quad E(M')$$

Therefore $E_{112}(M) \not\approx_{obs} E_{112}(M')$.
Thus, C_{obs} distinguish $E_{11}(M)$ from $E_{11}(M')$ since $M \neq M'$.

6.2. Example 2: Authenticity attack

Recall the second example. The protocol could stated as follow,

$$\textbf{Message 1} : A \rightarrow B : K_A(M)$$

$$\textbf{Message 2} : B \rightarrow A : K_B(M)$$

The rewrite rules of the protocol are:

$$\left[\begin{array}{l} wait(0,x_1,A,x_2)\rightarrow \\ send(1,A,A,B,K_A(M))\cdot wait(2,B,A,K_B(x_M)) \end{array}\right] \quad (R1)$$

$$\left[\begin{array}{l} send(1,A,A,B,K_A(M))\cdot wait(1,x_3,B,x_4)\rightarrow \\ send(2,B,B,A,K_B(M))\cdot wait(1,x_3,B,x_4) \end{array}\right] \quad (R2)$$

$$\left[\begin{array}{l} send(2,B,B,A,K_B(M))\cdot wait(2,B,A,K_B(x_M))\rightarrow \\ wait(0,x_1,A,x_2) \end{array}\right] \quad (R3)$$

The execution of the protocol is as follows:
$E_1 = wait(0, x_1, A, x_2) \cdot wait(1, x_3, B, x_4)$
$\rightsquigarrow$
$E_{11} = send(1, A, A, B, K_A(M)) \cdot wait(2, B, A, K_B(x_M)) \cdot wait(1, x_3, B, x_4)$
$\rightsquigarrow$
$E_{111} = send(1, B, B, A, K_B(M)) \cdot wait(1, x_3, B, x_4) \cdot wait(2, B, A, K_B(x_M))$
$\rightsquigarrow$
$E_{1111} = wait(1, x_1, A, x_2) \cdot wait(1, x_3, B, x_4)$

Let's consider the following instance of the protocol:

$$\textbf{Message 1} : A^{spec} \rightarrow B : K_A(M)$$

$$\textbf{Message 2} : B \rightarrow A^{spec} : K_B(M)$$

The running of a session of this specification is the same as that of the protocol, however A^{spec} is passed for A when he sends any message. For example, the node 1:1 changes as follow:
E_{11}
$\rightsquigarrow$
$E_{112} = E^{spec} = send(x, A^{spec}, A, B, K_A(M)) \cdot wait(1, x_3, B, x_4) \cdot wait(2, B, A, K_B(x_M))$
To prove the attack, one has to prove:

$$\exists\ i\ \in N^+ E_i \not\approx_{obs} {E_i}^{spec} \quad (1)$$

First step: we have to prove:

$$E_{112} \not\approx_{obs} {E_{112}}^{spec} \quad (2)$$

Let S et S^{spec} stated as follow:

$$S^{spec} = send(x, A^{spec}, A, B, K_A(M))$$
$$\text{S} \quad = send(x, A, A, B, K_A(M))$$

Since $K_A(M) \approx_{obs} K_A(M)$ et $A \not\approx_{obs} A^{spec}$, S $\not\approx_{obs} S^{spec}$
Therefore, the authenticity property is not verified.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

6.3. Example 3: Secrecy protected

This example exchanges messages encrypted with the public key K between two principles A and B. The protocol is simply described as follows: A sends a secret message M under the public key K, i.e $K(M)$ to B. Then, B send the same message under the public key K, i.e $K(M)$ when he receives the message **M** from A. This is specified as follows:

$$\textbf{Message 1} : A \rightarrow B : K(M)$$

$$\textbf{Message 2} : B \rightarrow A : K(M)$$

The rewrite rules of the protocol are:

$$\left[\begin{array}{l} wait(0,x_1,A,x_2)\rightarrow \\ send(1,A,A,B,K(M))\cdot wait(2,B,A,K(x_M)) \end{array}\right] \quad (R1)$$

$$\left[\begin{array}{l} send(1,A,A,B,K(M))\cdot wait(1,x_3,B,x_4)\rightarrow \\ send(2,B,B,A,K(M))\cdot wait(1,x_3,B,x_4) \end{array}\right] \quad (R2)$$

$$\left[\begin{array}{l} send(2,B,B,A,K(M))\cdot wait(2,B,A,K(x_M))\rightarrow \\ wait(0,x_1,A,x_2) \end{array}\right] \quad (R3)$$

The execution of the protocol is as follows:
$E_1 = wait(0, x_1, A, x_2) \cdot wait(1, x_3, B, x_4)$
$\rightsquigarrow$
$E_{11} = send(1, A, A, B, K(M)) \cdot wait(2, B, A, K(x_M)) \cdot wait(1, x_3, B, x_4)$
$\rightsquigarrow$
$E_{111} = send(2, B, B, A, K(M)) \cdot wait(1, x_3, B, x_4) \cdot wait(2, B, A, K(x_M))$
$\rightsquigarrow$
$E_{1111} = wait(1, x_1, A, x_2) \cdot wait(1, x_3, B, x_4)$

Let

$$send(x, I, A, B, C[\_])$$

be the observable context (that an intruder can apply on the intercepted message), where:

- I: Intruder name,
- A: sender name,
- B: receiver name,
- C[_]: a context

So, the node 1:1:1 changes as follows:
E_{11}
$\rightsquigarrow$
$E_{112} = send(x, I, A, B, C[K(M)]) \cdot wait(1, x_3, B, x_4) \cdot wait(2, B, A, K(x_M))$
Besides, it is clear that during the first and the second step of the protocol, A and B behave in the same way. As a consequence, it is quite enough to show that the secret is protected in the first step so that it will be in the second step. We conclude that to prove secrecy property, one must prove the equation 3:

$$\forall M, M',\ E_{112}(M) \approx_{obs} E_{112}(M') \qquad (3)$$

Let,

E(M) = $send(x, I, A, B, C[K(M)])$
E(M') = $send(x, I, A, B, C[K(M')])$

According to lemma 5.1, $K(M) \approx_{obs} K(M')$. Therefore, $\forall\ c \in C_{obs}\ c[K(M)] = c[K(M')]$. Second, it is obvious that $B \approx_{obs} B$ and $A \approx_{obs} A$. Thus, $E(M) \approx_{obs} E(M')$. Finally, using congruence, $E_{112}(M) \approx_{obs} E_{112}(M')$. Therefore the secret is protected.

7. Conclusion

In this paper, we gave ideas on how to reason about cryptographic protocols in the framework of observational theories. We showed how to express security properties in terms of observational equivalence. We express secrecy properties as simple equations that mean indistinguishability in the presence of an arbitrary attacker.

It will be important to think about how to schematize, in a finite way, all the observable contexts representing the set of actions that an intruder can do on intercepted message. Berregeb et al. [8] presented an automated procedure for proving observational properties in conditional specifications. The method relies on the construction of a set of a *critical contexts* which enables to have the completeness. However, Berregeb et al. [9] construct sets of cover-contexts that enable to prove observational properties in the case of equational systems. Such work could have an important contribution in the automation of our approach.

Acknowledgements : we would like to thank Yassine LAKHNECH for valuable discussions. We would like also to thanks the referees for valuable comments.

References

[1] M. Abadi, C. Fournet, *Mobile values, new names, and secure communication*, In Proceedings of the 28th ACM Symposium on Principles of Programming Languages (POPL'01), pages 104-115, January 2001.

[2] M. Abadi, A. D. Gordon, *A calculus for cryptographic protocols : The spi calculus*, In proceedings of the 4th ACM Conference on Computer and Com munications Security, pages 36-47, ACM Press, 1997.

[3] M. Abadi, A. D. Gordon, *A calculus for cryptographic protocols: The spi-calculus*, In Fourth ACM Conference on Computer and Communication Security, pages 36-47, ACM Press, 1999.

[4] R. Amadio, D. Lugiez, *On the reachability problem in cryptographic protocols*, In Proceedings of the 12th International Conference on Concurrency Theory (CONCUR'00), 2000.

[5] R. Amadio, D. Lugiez, V. Vanackère, *On the symbolic reduction of processes with cryptographic functions*, Theoretical Computer Science, 290(1) :695-740, 2002.

[6] G. Bernot, M. Bidoit, T. Knapik, *Behavioural approaches to algebraic specifications: A comparative study*, Acta Informatica, 31(7):651-671, 1994.

[7] G. Bernot, M. Bidoit, T. Knapik *Behavioural Theories*, In Recent Trends in Data Type Specification, Volume 906 of Lecture Notes in Computer Science, Pages 153-169, 1995.

[8] N. Berregeb, A. Bouhoula, M. Rusinowitch, *Observational proofs with critical contexts*, Fundamantal Approches to Software Engeneering (ETAP 98)LNCS 1382, pp38-53, March/April 1998.

[9] N. Berregeb, R. Robbana, A. Tiwari, *Towards automated proofs of observational properties*, Discrete Mathematics and theoretical Computer Science 6, pp143-162, 2004.

[10] M. Bidoit, R. Hennicker *On the inegration of observability and reachability concepts*, In Proc. 5th Int. Conf. Foundations of Software Science and Computation Structures (FOSSACS2002), Grenoble, France, April 2002.

[11] M. Boreale, R. De Nicola, R. Pugliese, *Proof techniques for cryptographic processes*, In Logic in Computer Science, pages 157-166, 1999.

[12] J. A. Clark, J. L. Jacob, *A survey of authentication protocol literature: Version 1.0.*, http ://www.cs.york.ac.uk/ jac/ papers/drareviewps.ps, 1997, November 1997.

[13] D. Dolev, A.C. Yao, *On the security of public key protocols*, In Proceedings of the 22nd Symp. on Foundations of ComputerScience, pages 350-357, Nashville, Tennessee, USA, 1981. IEEE Computer Society Press.

[14] H. Ehrig, B. Mahr, *Fundamentals of Algebraic Specifications*, ETACS Monographs on Theoretical Computer Science, Vol 6, Springer-Verlag, 1985.

[15] R. Hennicker, M. Bidoit, A. Kurz *Observational logic, constructor-based logic, and their duality*, Theorotical computer science, 298(3):471-510, 2003.

[16] F. Jacquemard , M. Rusinowitch , L. Vigneron, *Compiling and Verifying Security Protocols*, In Technical Report 3938 , Centre de Recherche en Informatique de Nancy Vandoeuvre-*lés*-Nancy, (May 30, 2000).

[17] J. Jürjens, *Model-Based Security Engineering with UML*, Springer Berlin / Heidelberg, Volume 3655/2005, pages 42-77, 2005.

[18] B.Schneir., *Applied Cryptography: Protocls, Algorithms, and Source Code in C*, John Wiley & Sons,Inc., second edition, 1996.

[19] S. Schneider, *Verifying authentication protocols with CSP*, In Proceedings of the 10th Computer Security Foundations Workshop (CSFW'97), Rockport, Massachusetts, USA, 1997. IEEE Computer Society Press.

IPOG: A General Strategy for T-Way Software Testing

Yu Lei[1], Raghu Kacker[2], D. Richard Kuhn[2], Vadim Okun[2], James Lawrence[3]

[1]Dept. of Comp. Sci. and Eng.
University of Texas at Arlington
Arlington, TX 76019
Email: ylei@cse.uta.edu

[2]Information Technology Laboratory
National Inst. of Standards and Tech.
Gathersburg, MD 20899
Email: {raghu.kacker, kuhn, vadim.okun}@nist.gov

[3]Dept. of Mathematics
George Mason University
Fairfax, VA 22030
Email: lawrence@gmu.edu

Abstract

Most existing work on *t*-way testing has focused on 2-way (or pairwise) testing, which aims to detect faults caused by interactions between any two parameters. However, faults can also be caused by interactions involving more than two parameters. In this paper, we generalize an existing strategy, called In-Parameter-Order (IPO), from pairwise testing to *t*-way testing. A major challenge of our generalization effort is dealing with the combinatorial growth in the number of combinations of parameter values. We describe a *t*-way testing tool, called FireEye, and discuss design decisions that are made to enable an efficient implementation of the generalized IPO strategy. We also report several experiments that are designed to evaluate the effectiveness of FireEye.

1. Introduction

One approach to software testing is combinatorial testing, which creates test suites by selecting values for input parameters and by combining these parameter values. For a system with *n* parameters, each of which has *d* values, the number of all possible combinations of values of these parameters is d^n. Due to resource constraints, it is nearly always impossible to exhaustively test all of these combinations of parameter values. Thus, a strategy is needed to select a subset of these combinations. One such strategy, called *t*-way testing, requires every combination of any *t* parameter values to be covered by at least one test, where *t* is referred to as the strength of coverage and usually takes a small value. The notion of *t*-way testing can substantially reduce the number of tests. For example, a system of 20 parameters that have 10 values each requires 10^{20} tests for exhaustive testing, but as few as 180 tests for 2-way (or pairwise) testing [6]. We can consider each combination of parameter values to represent one possible interaction among these parameters. The rationale behind *t*-way testing is that not every parameter contributes to every fault, and many faults can be exposed by interactions involving only a few parameters.

To illustrate the concept of *t*-way testing, consider an elementary software system consisting of three Boolean parameters. Denote the two values of a Boolean parameter as 0 and 1. Fig. 1 shows a pairwise test set for this system. In the test set, each row represents a test, and each column represents a parameter (in the sense that each entry in a column is a value of the parameter represented by the column). It can be checked that each of the three pairs of columns, i.e., columns 1 and 2, columns 1 and 3, and columns 2 and 3, contains all four pairs of values of two Boolean parameters, i.e., {00, 01, 10, 11}. If all failures of the system are triggered by faulty interactions between at most two parameters, this test set would allow all the failures to be detected. Note that an exhaustive test set for this system would consist of $2^3 = 8$ tests.

$$\begin{pmatrix} 0 & 0 & 0 \\ 0 & 1 & 1 \\ 1 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix}$$

Figure 1. A 2-way test set for 3 boolean parameters

Existing work on *t*-way testing has mainly focused on pairwise testing, which aims to detect faults that are caused by interactions between any two parameters. However, faults can also be caused by interactions involving more than two parameters [10][11]. In order to effectively detect those faults, it is necessary to enable a higher strength of coverage. In this paper, we generalize an existing strategy, called In-Parameter-Order (or IPO), from pairwise testing to general *t*-way testing. The resulting strategy is referred to as In-Parameter-Order-General (or IPOG). A major challenge of our generalization effort is dealing with the combinatorial growth in the number of combinations of parameter-values. We describe a *t*-way testing tool called FireEye, and discuss design decisions that are made to enable an efficient implementation of the IPOG strategy. We also report several experiments that were conducted to evaluate the effectiveness of FireEye. In particular, we conducted an experiment that compared FireEye to several existing tools. The result of this experiment indicates that FireEye performed significantly better than the other tools for a real-life application.

The remainder of the paper is organized as follows. Section 2 briefly reviews existing work on *t*-way testing. Section 3 describes the IPOG strategy. An algorithm that implements the IPOG strategy is also presented in Section 3. Section 4 describes the FireEye tool, and discusses several key design decisions. Section 5 reports the design

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

and the results of the experiments. Section 6 provides concluding remarks and our plan for further work

2. Related Work

Cohen et al. proposed a strategy, called Automatic Efficient Test Generator (or AETG), which constructs a test set by repeatedly adding one test at a time until all the combinations of parameter values are covered [4][5]. A greedy algorithm is used to construct the tests such that each test covers as many uncovered combinations as possible. Several variants of this strategy have been reported in the literature [2][17]. These variants share the same framework as AETG but use different heuristics for the greedy construction of each test [6]. In [13][16], we proposed the IPO strategy, which builds a pairwise test set for the first two parameters, extends the test set to cover the first three parameters, and continues to extend the test set until it builds a pairwise test set for all the parameters. Covering one parameter at a time allows the IPO strategy to achieve a lower order of complexity than AETG. Most recently, heuristic search techniques such as hill climbing and simulated annealing have been applied to multi-way testing [6]. Unlike AETG and IPO, which builds a test set from scratch, heuristic search techniques start from a pre-existing test set and then apply a series of transformations to the test set until a test set is reached that covers all the combinations. Heuristic search techniques can produce smaller test sets than AETG and IPO, but they typically take longer to complete.

In addition to computational approaches, algebraic approaches have also been reported. These approaches construct test sets using pre-defined rules. Some algebraic approaches compute test sets directly by a mathematical function. These approaches are generally extensions of the mathematical methods for constructing orthogonal arrays [1][14]. Informally, an orthogonal array of strength t requires that every possible combination of any t columns be covered exactly once. Therefore, an orthogonal array can be considered as an optimal t-way test set if we consider each row to represent a test and each column to represent a parameter. Other algebraic approaches are based on the idea of recursive construction, which allows larger test sets to be constructed from smaller ones [8][18].

Computational and algebraic approaches have their own advantages and disadvantages. Computational approaches can be applied to arbitrary system configurations, but they can be expensive as they involve explicit enumeration and there can be a large number of combinations to be enumerated. The computations involved in algebraic approaches are typically lightweight, and in some cases, algebraic approaches can produce optimal test sets. However, algebraic approaches often impose restrictions on the system configurations to which they can be applied.

Finally, many empirical studies have been reported on assessing the fault detection effectiveness of t-way testing. In [3], Burr and Young showed that pairwise testing achieves higher block and decision coverage than traditional methods for a commercial email system. In [7], Dalal et al. applied t-way testing to a telephone software system and showed that several faults can only be detected under certain combinations of input parameters. In [10][11], Kuhn et al. studied the actual faults in several software projects, and found that all the known faults are caused by interactions among 6 or fewer parameters.

3. The IPOG Strategy

In this section, we present the IPOG strategy. Our motivation for generalizing the IPO strategy is two-fold. First, we want to develop a testing strategy that can be applied to general software applications. Thus, the strategy should put no restrictions on the system configuration under test. This consideration favors computational approaches over algebraic approaches. (Recall from Section 2 that the former can be applied to an arbitrary system configuration, while the latter often has restrictions on the system configurations to which they can be applied.) Second, general t-way testing has a more stringent demand on the time and space requirements than pairwise testing. This is because the number of combinations grows exponentially as the strength of coverage increases. This consideration favors the IPO strategy over other strategies such as AETG and heuristic search techniques. We also note that the IPO strategy is deterministic, i.e., it always produces the same test set for the same system configuration.

The framework of the IPOG strategy can be described as follows: For a system with t or more parameters, the IPOG strategy builds a t-way test set for the first t parameters, extends the test set to build a t-way test set for the first t + 1 parameters, and then continues to extend the test set until it builds a t-way test set for all the parameters. (The parameters can be in an arbitrary order.) The extension of an existing t-way test set for an additional parameter is done in two steps:

- *horizontal growth*, which extends each existing test by adding one value for the new parameter;
- *vertical growth*, which adds new tests, if needed, to the test set produced by horizontal growth.

Fig. 2 shows a test generation algorithm called IPOG-Test that implements this framework. The algorithm takes as input an integer t and a set ps of parameters, and produces as output a t-way test set for the parameters in set ps. It is assumed that the number n of parameters in set ps is greater than or equal to t. Fig. 3 shows an application of algorithm IPOG-Test to an example system for 3-way testing. This

```
Algorithm IPOG-Test (int t, ParameterSet ps)
{
1.  initialize test set ts to be an empty set
2.  denote the parameters in ps, in an arbitrary order, as P1, P2, ..., and Pn
3.  add into test set ts a test for each combination of values of the first t parameters
4.  for (int i = t + 1; i ≤ n;  i ++){
5.     let π be the set of t-way combinations of values involving parameter Pi
          and t -1 parameters among the first i – 1 parameters
6.     // horizontal extension for parameter Pi
7.     for (each test τ = (v1, v2, ..., vi-1) in test set ts) {
8.        choose a value vi of Pi and replace τ with τ' = (v1, v2, ..., vi-1, vi) so that τ' covers the
             most number of combinations of values in π
9.        remove from π the combinations of values covered by τ'
10.    }
11.    // vertical extension for parameter Pi
12.    for (each combination σ in set π){
13.       if (there exists a test that already covers σ) {
14.          remove σ from π
15.       } else {
16.          change an existing test, if possible, or otherwise add a new test
             to cover σ and remove it from π
17.       }
18.    }
19.}
20.return ts;
}
```

Figure 2: Algorithm IPOG-Test

example system consists of four parameters P1, P2, P3, and P4, where P1, P2, P3 have two values 0 and 1, and P4 has three values 0, 1, and 2. In the following, we will use this application as a running example to explain how algorithm IPOG-Test works.

Algorithm IPOG-Test begins by initializing test set *ts* to be empty (line 1), and by putting the input parameters into an arbitrary order (line 2). Note that test set *ts* will be used to hold the resulting test set. Next, the algorithm builds a *t*-way test set for the first *t* parameters. This is trivially done by adding into test set ts a test for every combination of the first t parameters (line 3). In Fig. 3, the 3-way test set built for the first three parameters is shown in part (a), which contains all the 8 possible combinations of the first three parameters, i.e., P1, P2, and P3.

If the number *n* of parameters is greater than the strength *t* of coverage, the remaining parameters are covered, one at each iteration, by the outermost for-loop (line 4). Let P*i* be the parameter that the current iteration is trying to cover. We first compute the set π of combinations that must be covered in order to cover parameter P_i (line 5). Covering parameter P*i* means extending test set *ts* so that it becomes a *t*-way test set for parameters $P_1, \ldots, P_{i-1}$, and P_i. Note that test set *ts* is already a *t*-way test set for parameters $P_1, \ldots, P_{i-1}$. Thus, we only need to cover all the *t*-way combinations involving P_i and any group of *t* -1 parameters among $P_1, \ldots, P_{i-1}$, which are the parameters that are already covered. For example, in Fig. 3, in order to cover P4, we need to cover all the 3-way combinations of the following parameter groups, (P1, P2, P4), (P1, P3, P4), and (P2, P3, P4). We will not list each of the combination in those parameter groups, as they can easily be enumerated. Instead, we only point out that each of these groups has 12 combinations. Thus, there are in total 36 combinations in the set π computed for Fig. 3.

The combinations in set π are covered in the following two steps:

- Horizontal growth: This step extends each of the existing tests by adding a value for parameter Pi (lines 7 - 10). These values are chosen in a greedy manner, i.e., at each step, the value chosen is one that covers the largest number of combinations in set π (line 8). Each time a value is added, the set of combinations covered due to this addition are removed from set π (line 9). For example, in Fig. 3, the 4th test is extended by adding the value 0 for P4, which covers three combinations in set π: {(P1.0, P2.1, P4.0), (P1.0, P3.1, P4.0), (P2.1,

P3.1, P4.0)}. Here notation Pi.v indicates that v is a value of parameter Pi. Note that if the 4th test was extended by adding the value 1 for P4, it would only cover two combinations in set π: {(P1.0, P2.1, P4.1), (P2.1, P3.1, P4.1)}. The reason is that the combination (P1.0, P3.1, P4.1) was covered by the 2nd test and thus was removed from set π when the 2nd test was extended.

- Vertical growth: This step covers the remaining uncovered combinations, one at a time, either by changing an existing test or by adding a new test (line 16). When we change a test to cover a combination, only *don't care* values can be changed. A *don't care* value is a value that can be replaced by any value without affecting the coverage of a test set. If no existing test can be changed to cover σ, a new test needs to be added in which the parameters involved in σ are assigned the same value in σ and the other parameters are assigned *don't care* values. For example, in Fig. 3, after horizontal growth, combination (P1.1, P2.0, P4.0) has not been covered yet. No existing test can be found such that it can be changed to cover this combination. Thus, we create a new test (P1.1, P2.0, P3.-, P4.0), which is the 9th test in part (c), to cover this combination, where "–" denotes a don't care value. Also note that (P2.0, P3.1, P4.0) is another combination that was not covered either after horizontal growth. This combination can be covered by changing the value of P3 from "-" to 1 in the 9th test.

Now we consider the complexity of algorithm *IPOG-Test*. The space complexity is dominated by the storage of π (line 5) for covering each new parameter. Let n be the number of parameters and d the largest domain size of the parameters. The space requirement for π is $O\left(d^t \times n^{t-1}\right)$. The time complexity is dominated by horizontal extension. In Section 4, we describe a data structure for storing all the combinations. With this data structure, it takes $O(1)$ time to determine whether or not a t-way combination is already covered, and it takes $O\left(n^{t-1}\right)$ time to determine the number of combinations covered by a test. Thus, it takes $O\left(d \times n^{t-1}\right)$ to determine which value of the new parameter covers the most t-way combinations. As shown in [5] and supported by the experiments in Section 5, the number of tests generated by algorithm IPOG-Test is in $O\left(d^t \times \log n\right)$. Thus, the time complexity of horizontal extension, and that of the entire algorithm, is $O\left(d^{t+1} \times n^{t-1} \times \log n\right)$.

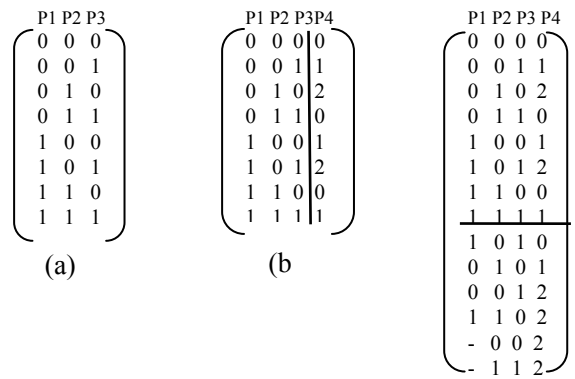


Figure 3. An illustration of algorithm IPOG-Test

4. FireEye: A T-Way Testing tool

We built a t-way testing tool, called FireEye, which implements the IPOG strategy. FireEye is written in Java and consists of the following major components: (1) *CombinatoricsHelper*, which is a utility class that is responsible for all the computations related to combinatorics; (2) *CombinationManager*, which manages the combinations in a way such that they can be stored and checked efficiently; (3) *TestEngine*, which implements algorithm IPOG-Test; (4) *TestGenerator*, which drives the entire test generation process. FireEye also provides a graphic user interface (GUI) to facilitate the use of this tool. The GUI allows the user to create, edit, and inspect system configurations, to set up runtime options, and to view the resulting test sets.

Due to the combinatorial effect, the number of t-way combinations can be large. To enable an efficient implementation, these combinations must be managed carefully. In Section 4.1, we discuss how FireEye computes t-way combinations. In Section 4.2, we describe the data structure for storing these combinations in FireEye.

4.1 Computing T-Way Combinations

In order to cover a new parameter, we first need to compute the set π of t-way combinations involving the new parameter and t-1 parameters that have already been covered (line 5 of Fig. 2). In the following, we consider a more general problem: How can we compute all n-way combinations of values of m parameters, where $n \le m$? Conceptually, this problem needs to be solved in two steps. First, we generate all possible combinations of n parameters out of m parameters. Second, for each combination of n parameters, we enumerate all possible combinations of values of these n parameters. In the

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

remainder of this paper, we will refer to a *combination of parameters* as a *parameter combination*, and a *combination of parameter values* as a *value combination*.

One approach to generating combinations of *n* elements is to use a nested loop of *n* levels, each iterating through the possible values of each element. This approach can be applied to generate both *n*-way parameter combinations, with care given to avoid generating the same combination of parameters in different orders, and *n*-way value combinations. This approach, however, suffers from the problem that such a nested loop must be hard-coded. As described below, FireEye uses a generic approach that allows parameter and value combinations to be generated without hard-coding any loops[1].

We first discuss how to generate parameter combinations. Center to our approach is the use of parameter vectors. A parameter vector has *m* dimensions, one for each parameter. Consider each parameter vector as representing a parameter combination as follows: Each dimension takes on a binary value, 0 or 1, which indicates whether the corresponding parameter is excluded or included, respectively, in the parameter combination. For example, assume that there are 5 parameters {P0, P1, P2, P3, P4}. Then a parameter vector 10101 represents a parameter combination {P0, P2, P4}. Thus, the problem of generating all the *n*-way parameter combinations is transformed to the problem of generating all the parameter vectors in which the number of 1s is exactly *n*.

One naïve approach to solving the above problem is to enumerate all possible parameter vectors of *m* dimensions, and then filter out those in which the number of 1s is not *n*. This enumeration can be accomplished as follows. Consider each vector to represent a numeric value, where each dimension represents a digit whose base is 2 and the significance of the digits decreases from left to right. Starting from a vector of all 0s, whose numeric value is 0, we can enumerate all the parameter vectors by repeatedly adding 1 until a vector of all 1s is reached. The addition of 1 to a vector can be done by setting the least significant digit *g* whose value is 0 to 1 and changing all the digits that are less significant than *g* to 0. For example, let 10011 be a parameter vector. Observe that the third digit (from left) is the least significant digit whose value is 0. In order to add 1 to this vector, we change the third digit from 0 to 1, and set the last two digits to 0. Doing so results in a new vector 10100.

Instead of enumerating all possible parameter vectors and then filtering out invalid ones, FireEye implements a more efficient approach that only generates valid vectors, i.e., those in which the number of 1s is exactly *n*. The framework of our approach is similar to that of the naïve approach, except for the following two differences. First, we start from a parameter vector in which the least significant *n* digits are set to 1, instead of the vector of all 0s. For example, let $m = 5$, and $n = 3$. Then, we start from 00111. Note that such a parameter vector is the smallest one, in terms of its numeric value, that consists of three 1s. Second, every time we derive a new parameter vector, we ensure that the number of 1s in the current vector is preserved. There are two cases to consider, depending on whether the last digit in the vector is 1 or 0.

- *Case 1*: If the last digit is 1, we find the least significant digit *g* that is 0 and is followed by 1. Then, we change *g* from 0 to 1 and the digit following g from 1 to 0. For example, assume that the current vector is 01011. Then, the third digit (from left) is the least significant digit that is 0 and is followed by 1. Thus, we generate the next parameter vector by changing the third digit from 0 to 1 and the fourth digit from 1 to 0, which produces 01101. Note that this new vector is the smallest one that is greater than the current vector, in terms of their numeric values, and that preserves the same number of 1s.
- *Case 2*: If the last digit is 0, we find the least significant digit *g* that is 0 and is followed by 1, which is similar to Case 1. At the same time, we count the number of 1s, say *c*, that appear before *g*. Then, we change *g* from 0 to 1, and set the digits that are less significant than g to 0, except for the last $n - c - 1$ digits, which are set to 1. For example, assume that the current vector is 10110. Then, the second digit (from left) is the least significant digit that is 0 and is followed by 1. Since the first digit is 1, $c = 1$. Thus, we generate the next parameter vector by changing the second digit from 0 to 1, and by setting the third and fourth digits to 0, and the last digit to 1, which results in 11001. Note that this new vector is the smallest one that is greater than the current vector, in terms of their numeric values, and that preserves the same number of 1s.

Next we discuss how to enumerate all possible value combinations for each parameter combination. Similar to the way we consider a parameter combination, we consider each value combination to represent a numeric value, where each dimension represents a digit whose base is the same as the domain size of the corresponding parameter and the significance of the digits decreases from left to right. Starting from a value combination of all 0s, whose numeric value is 0, we can enumerate all the value combinations by repeatedly adding 1 until we reach a value combination in which the value of each digit is its base

[1] We developed the described approach independently, but were made aware of [15] that provides a similar solution in the review process.

minus 1. The addition of 1 to a value combination can be accomplished by incrementing the least significant digit *g* whose value is less than its base minus 1 and setting all the digits that are less significant than *g* to 0. For example, assume that there are three parameters P1, P2, and P3, each having three values. Let 112 be a value combination of the three parameters. The second digit is the least significant digit whose value is less than its base minus 1. We can add 1 to this combination by incrementing the second digit and by setting the last digit to 0, which results in a new value combination 120.

4.2 Storing T-Way Combinations

In this section, we describe the data structure used by FireEye for storing *t*-way combinations. On the one hand, we want the storage to be as compact as possible. On the other hand, we want to be able to quickly determine whether or not a given combination has been covered, which is the most frequently performed operation in algorithm IPOG-Test.

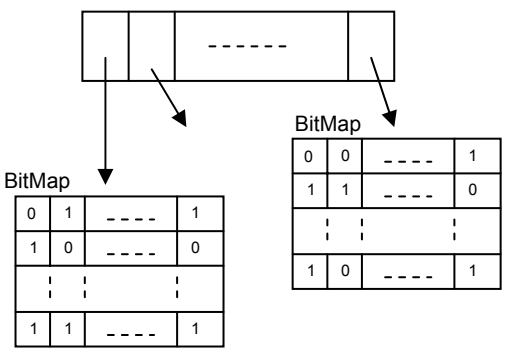


Figure 4. A two-level hierarchy for storing combinations

As shown in Fig. 4, the data structure is a hierarchy of two levels. At the first level is an array of pointers, each of which represents one possible parameter combination and points to a bitmap at the second level. The pointers are indexed in such a way that for a given parameter combination, we can directly compute its index and thus locate the corresponding pointer quickly without having to search through the array. We use an example to illustrate the indexing scheme. Assume that there are 4 parameters, P0, P1, P2, and P3. There are 4 combinations of 3 parameters out of the 4 parameters, and we index them in the following order: (P0, P1, P2), (P0, P1, P3), (P0, P2, P3), and (P1, P2, P3). The index of a given parameter combination (P_i, P_j, P_k) can be computed using the following formula $3 \times i + 2 \times (j - i - 1) + (k - j - 1)$. For instance, the index of (P0, P2, P3) is $3 \times 0 + 2 \times (2 - 0 - 1) + (3 - 2 - 1) = 2$. This formula can be easily generalized to any number of parameters.

At the second level, each bitmap has one bit for each value combination. The bit value 0 indicates that the corresponding value combination has not been covered yet, and the value 1 indicates the corresponding value combination has already been covered. Again, we consider each value combination to represent a numeric value. The numeric value of a value combination is used to index the bit that corresponds to the combination.

In order to determine whether or not a given value combination is covered, we first find the pointer that points to the bitmap to which the value combination belongs. Then, we check the value of the bit corresponding to the combination in the bitmap. Both steps take constant time.

5. Experimental Results

Our experiments have two goals. First, we want to study the growth in the size of the test sets generated by algorithm IPOG-Test, as well as the time taken to produce those test sets, in terms of the strength of coverage, the number of parameters, and the domain size, respectively. Second, we want to compare the performance of FireEye to existing tools, both in terms of the size of the resulting test sets and the time taken to produce these test sets.

To accomplish the first goal, we applied FireEye to three series of system configurations. In the first series, the number of parameters is fixed to 10, the domain size of each parameter is fixed to 5, and the strength of coverage is varied from 2 to 6. In the second series, the strength of coverage is fixed to be 4, the domain size of each parameter is fixed to be 5, and the number of parameters is varied from 5 to 15. In the third series, the strength of coverage is fixed to be 4, the number of parameters is fixed to be 10, and the domain size is varied from 2 to 10.

Tables 1, 2 and 3 show the experimental results for the three series of system configurations, respectively. The columns in the three tables are self-explanatory. Note that the execution times are shown in seconds, and all the results were collected using a laptop running Windows XP with 1.6GHZ CPU and 1GB memory.

In [5], it was shown that the growth in the size of a test set is in $O(d^t \log n)$, where *t* is the strength of coverage, *d* is the domain size, and *n* is the number of parameters. We performed curve fitting analysis on the sizes of the test sets in the three tables. The analysis showed that our experimental results were consistent with the theoretical results. In particular, we note that the number of tests in a *t*-way test grows very quickly as the strength of coverage *t* increases.

To accomplish the second goal, we identified the following existing tools that support *t*-way testing and are either open source or free for academic use: (1) Intelligent Test Case

Handler (or ITCH), which is from IBM [19]; (2) Jenny, which is from www.burtleburtle.net [20]; (3) TConfig, which is from University of Ottawa [21]; and (4) Test Vector Generator (or TVG), which is from www.SourceForge.com [22]. Based on limited information available in the literature, ITCH implements a combination of several algebraic methods (the details of the combination are not known), and TConfig implements a recursive construction method. Both Jenny and TVG seem to implement a computational method, but the details of their algorithms are not clear. Note that all these tools are written in Java, except for Jenny, which is written in C.

t-way	2	3	4	5	6
Size	48	308	1843	10119	50920
Time	0.11	0.56	6.38	63.8	791.35

Table 1: Results for 10 5-value parameters for 2- to 6-way testing

We applied FireEye and the above tools to a Traffic Collision Avoidance System (TCAS) module. It implements part of an aircraft collision avoidance system specified by the Federal Aviation Administration, and has been used in other studies of software testing [9][12]. The TCAS module has twelve parameters: seven parameters have 2 values, two parameters have three values, one parameter has four values, and two parameters have 10 values. Table 4 shows the sizes of the test sets generated by each tool and the times taken to generate these test sets. The execution times are shown in seconds, if not specified otherwise. The sizes of some test sets are not available, shown as NA, as their construction seems to take an excessive amount of time. In all cases, FireEye has performed better than the other tools, both in terms of the sizes of the test sets and the execution times. In several cases, FireEye has performed substantially better, especially for 5- and 6-way testing. If we compare FireEye to a particular tool, the extent to which FireEye outperformed increases as the strength of coverage increases.

6. Conclusion and Future Work

We consider *t*-way testing to be a very promising testing technique for several reasons. First, as a specification-based technique, it requires no knowledge about the implementation under test. Moreover, the specification required by *t*-way testing is lightweight, as a basic system configuration only needs to identify the input parameters and the possible values of each of those parameters. Second, *t*-way testing can be very effective for various types of applications. Kuhn et al. studied actual faults in several industrial applications, showing that all the known faults in these applications are caused by up to 6-way interactions [11]. Finally, test input generation for *t*-way testing can be automated as a push-button feature, which is a key to industrial acceptance.

# of params	5	6	7	8	9	10	11	12	13	14	15
Size	784	1064	1290	1491	1677	1843	1990	2132	2254	2378	2497
Time	0.19	0.45	0.92	1.88	3.58	6.38	10.83	17.52	27.3	41.71	61.26

Table 2: Results for 5 to 15 5-value parameters for 4-way testing

# of values	2	3	4	5	6	7	8	9	10
Size	46	229	649	1843	3808	7061	11993	19098	28985
Time	0.16	0.547	1.8	6.33	16.44	38.61	83.96	168.37	329.36

Table 3: Results for 10 parameters with 2 to 10 values for 4-way testing

t-way	FireEye		ITCH		Jenny		TConfig		TVG	
	Size	Time	Size	Time	Size	Time	Size	Time	Size	Time
2	100	0.8	120	0.73	108	0.001	108	>1 hour	101	2.75
3	400	0.36	2388	1020	413	0.71	472	>12 hour	9158	3.07
4	1361	3.05	1484	5400	1536	3.54	1476	>21 hour	64696	127
5	4219	18.41	NA	>1 day	4580	43.54	NA	>1 day	313056	1549
6	10919	65.03	NA	>1 day	11625	470	NA	>1 day	1070048	12600

Table 4: Results of different tools for the TCAS configuration

We are continuing our work in the following directions. First, the IPOG strategy needs to explicitly enumerate all possible combinations. When the number of combinations is large, explicit enumeration can be prohibitive. We are developing techniques to reduce the number of combinations that are enumerated. Second, we are extending algorithm IPOG-Test to support parameter relations and constraints. Parameter relations are used to avoid exercising combinations between parameters that do not interact with each other. Parameter constraints are used to exclude combinations that are not meaningful from the domain semantics. Finally, *t*-way testing can generate a large number of tests, which makes it impractical to manually execute the tests and evaluate their results. We plan to integrate our test generation tool with other tools to automate the entire testing process, i.e., including test generation, test execution, and test evaluation.

Acknowledgement

The authors would like to thank Profs. Renee Bryce and Richard Carver for their comments on an earlier version of this paper. The authors would also like to thank Chinmay Jayaswal for conducting the experiments. This paper is also greatly improved by the anonymous review comments. This work is partly supported by a grant (Award No. 60NANB6D6192) from the Information Technology Lab (ITL) of National Institute of Standards and Technology (NIST).

Disclaimer: Certain software products are identified in this document. Such identification does not imply recommendation by NIST, nor does it imply that the products identified are necessarily the best available for the purpose.

References

[1] K. A., Bush, "Orthogonal arrays of index unity," Annals of Mathematical Statistics, 23 (1952), 426-434.

[2] R. Bryce, A Deterministic Density Algorithm for Pairwise Interaction Coverage, Proceedings of the International Conference on Software Engineering (SE 2004). Innsbruck, Austria, pp. 245-252.

[3] K. Burr and W. Young, "Combinatorial test techniques: Table-based automation, test generation and code coverage," in *Proc. of the Intl. Conf. on Software Testing Analysis & Review*, 1998.

[4] D. M. Cohen, S. R. Dalal, J. Parelius, G. C. Patton, "The Combinatorial Design Approach to Automatic Test Generation," IEEE Software, Vol. 13, No. 5, pp. 83-87, September 1996.

[5] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton, "The AETG System: An Approach to Testing Based on Combinatorial Design," IEEE Transactions on Software Engineering, 23:7, 1997.

[6] M. B. Cohen, C. J. Colbourn, P. B. Gibbons and W. B. Mugridge, "Constructing test suites for interaction testing," In Proc. of the Intl. Conf. on Software Engineering, (ICSE 2003), 2003, pp. 38-48, Portland.

[7] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz, "Model-based testing in practice," in Proc. of the Intl. Conf. on Software Engineering, (ICSE), 1999, pp. 285–294.

[8] Alan Hartman, Leonid Raskin, "Problems and algorithms for covering arrays," Discrete Mathematics 284(1-3): 149-156 (2004)

[9] M. Hutchins, H. Foster, T. Goradia, and T. Ostrand. Experiments on the effectiveness of dataflow- and controlflow-based test adequacy criteria. In *Proc. Sixteenth Internat. Conf. On Software Engineering*, pp. 191–200, May 1994.

[10] D. R. Kuhn and M. J. Reilly, "An Investigation of the Applicability of Design of Experiments to Software Testing," Proceedings of the 27th NASA/IEEE Software Engineering Workshop, NASA Goddard Space Flight Center, December 2002.

[11] D. R. Kuhn, D. Wallace, A. Gallo, "Software Fault Interactions and Implications for Software Testing," IEEE Transactions on Software Engineering, June 2004, Vol. 30, No. 6.

[12] D. R. Kuhn, V. Okun, "Pseudo-exhaustive Testing For Software," 30th NASA/IEEE Software Engineering Workshop, April 25-27, 2006.

[13] Y. Lei and K. C. Tai , "In-parameter-order: a test generation strategy for pairwise testing," Proceedings of 3rd IEEE Intl. Conf. on High-Assurance Systems Engineering Symposium, 1998, pp. 254-261.

[14] R. Mandl, "Orthogonal Latin squares: an application of experiment design to compiler testing," Communications of the ACM, v.28 n.10, p.1054-1058, Oct. 1985.

[15] D. Stanton and D. White, Constructive Combinatorics, Springer, 1986.

[16] K. C. Tai and Y. Lei, "A Test Generation Strategy for Pairwise Testing," IEEE Transactions on Software Engineering, 2002, Vol. 28, No. 1.

[17] Y. W. Tung and W. S. Aldiwan, "Automating test case generation for the new generation mission software system," Proceedings of IEEE Aerospace Conference, 2000, pp. 431-437.

[18] A. W. Williams and R. L. Probert. A practical strategy for testing pair-wise coverage of network interfaces. In Proceedings of the 7th International Symposium on Software Reliability Engineering (ISSRE), White Plains, New York, 1996.

[19] ITCH, http://www.alphaworks.ibm.com/tech/whitch.

[20] Jenny, http://www.burtleburtle.net/bob/math/.

[21] TConfig, http://www.site.uottawa.ca/~awilliam/.

[22] TVG, http://sourceforge.net/projects/tvg/.

Model Checking of Computer-based Systems*

Jinzhao Wu[1,2]
1.College of opto-electronic Information
University of Electronic Science and Technology
Chengdu, China 610054
himrwujz@yahoo.com.cn

Wei Yan[2]
2.Chengdu Institute of Computer Applications
Chinese Academy of Sciences
Chengdu, China 610041
yanwei_bgm@126.com

Abstract

Computer-based systems, especially multi-agent systems which incorporate multi-agents physically distributed, can be represented as event structures. We propose an action-based event structure logic which is an appropriate approach specifying the action behaviors that cause states change. We discuss the symmetry of event structures. The symmetry reduction technique is used to reduce an event structures to get a simple quotient structure through the equivalence class induced by a permutation group. The action-based event structure formulas without confliction modality can be easily verified on the quotient structure for multi-agent systems.

***Keywords**: Computer-based systems, model checking, event structures, verification*

1. Introduction

Computer-based systems controlling from mobile phones to airplanes, have been used in many fields such as banks, governments and armis, undertaking the task of information collection, mathematical computation and weather forecasting. In many instances, errors in such systems can bring dramatic consequences. Hence, the demand and guarantee for error-free designs is very urgent. Naturally, as a response, the well-known methodology of model checking developed by Clarke and Emerson [2] makes much promise.

As we know, model checking has made great progress in finite-state concurrent systems. From a concurrent point of view, computer-based systems may be also a type of concurrent systems containing multi-components (or agents) physically distributed. However, there are something different between them need to pay attention. In most instances, action based models are more suitable for modelling of computer-based systems in contrast to widely applied state-based models of finite-state concurrent systems. The most common instantiations of action-based models are event structure which alleviate the state space explosion problem in system verification due to its partial ordered structure [18]. In this endeavor, event structures can be used as models, and the model checking of event structures provide guarantee for correctness of computer-based systems.

However, there are still two problems must be solved. First, in order to specify the property of computer-based systems, an action-based logic for event structures must be established. Second, even for a simple computer-based system, the event structure for it may be too complex and thus hard to used in model checking. In fact, there are many logic systems for event structures. For example, ESL and ESL[C].

The first logic dealing with n-sequential event structures called $SESL$ (Sequential Agent Event Structure Logic) was proposed by Lodaya and Thiagarajan [6] in 1987. Then Penczek presented a logic called ESL (Event Structure Logic) based on unrestricted event structures [16, 15]. Then many variant of event structure logic have been proposed to enhance the expressiveness. For example, $ESL[\delta]$ (ESL with a run proposition) can express the eventuality property, $DESL$ (Discrete Event Structure Logic) can reason about discrete event structures, and $ESL[C]$ (ESL with a concurrency operator) can capture the concurrency relation [9, 8, 15, 13]. Interpreted over labelled prime event structures, a temporal logic termed logic of causal knowledge was defined in [14]. In addition, first-order logic FOL and monadic second-order logic $MSOL$ were defined in [7] over regular trace event structures. These languages have been widely applied in computer science and other fields and have made great progress.

But, these logics are all purely state-based logics. The basic building blocks of these logics are state propositions, and therefore do not fit well to the specification of action-based behaviors which are important for many computer-

*This work is supported by National Natural Science Foundation of China under Grant No. 60373113 and by NKBRPC (No. 2004CB318000)

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

based systems.

Some action-based approaches have been developed to capture the reason for state transition. [11] is the first paper discussing how to transform the state-based branching-time logic CTL^* (Computation Tree Logic) to action-based CTL^*. Most of linear and branching-time temporal logics have action-based counterparts. See [10] for a review. Interpreted over prime event structures and based on pomset modalities, two partial-order logic systems were defined in [4, 17], which are used to characterize history preserving bisimulation on event structures. These two logics are action-based. However, the logics depend on individual event structure, and therefore are not general languages for specifying computer-based systems. We are unaware of any other event structure logic that is action-based.

In this paper, we propose an action-based event structure logic (for short, aESL[C]) which is actually an extension of event structure logic with concurrency (ESL[C]). We extend ESL[C] by modifying the set of atomic propositions which can be divided into two groups with respect to their properties. These are atomic state propositions and atomic action propositions. Atomic state propositions were inherited from atomic propositions of ESL[C], whereas atomic action propositions represent the reasons that actually cause state change. The logic describing in this paper can be used to specifying the relation between actions and states, and the interactive behaviors between agents.

Some special axioms corresponding to the interpretation of action propositions are added to original proof system. Then, we show that the axiom system is sound and complete. This is the first paper extending logics on event structure with action propositions. Due to its nice features, the logic can be used to capture the interactive behaviors of computer-based systems. We believe that the action-based event structure logic is a useful contribution to the area of reasoning about action executions of computer-based systems.

To solve the second problem, we introduce the symmetry reduction technique. The conception of permutation groups [3] is used to partition the prime traces of an event structure and thus to get a simple quotient structure. The quotient structure can be used to verify the properties except the confliction relation, since the quotient structure does not preserve the conflict property. We believe that the introduction of permutation groups to event structure is an useful step.

The rest of this paper is organized as follows: Section 2 simply introduces the basic conception of prime event structures. The action-based event structure logic aESL[C] is presented in Section 3. Section 4 discussed the symmetry reduction of event structures. Section 5 discusses the event structure and its quotient structure. The model checking is given in Section 6. The last section summarizes the paper.

2. Basic definitions of event structures

We start with the definition of event structures which we use throughout this paper. Only the basic concepts of prime event structures which were introduced firstly by [12] are sketched here.

Now, the main concepts of prime event structures [1] are introduced. Let Act be a set of actions, assume that each action has a unique name in Act.

Definition 2.1*(Prime Event Structure)*
A (labeled) prime event structure $\mathcal{E}$ is a 4-tuple $(E, \sharp, \preceq, l)$ with

- E, a set of events,
- $\sharp \subseteq E \times E$, is an irreflexive and symmetric relation, conflict relation,
- $\preceq \subseteq E \times E$, is a partial order, the causality relation.
- $l : E \rightarrow Act$, the action-labeling function.

such that for all event $e \in E$

1. $\{e' \mid e' \preceq e\}$ is finite,
2. $\forall e', e'' \in E : (e \sharp e' \wedge e' \preceq e'') \Rightarrow e \sharp e''$.

An event structure represents a computer-based system in the following way: the action names are in accordance with the activities which the system may perform, an event labelled $a \in Act$ stands for a particular occurrence of action a, $e \preceq e'$ denotes that in any computation, e' can occur only if e' has already occurred, $e \sharp e'$ means that e and e' can never occur together in the same run.

The first condition of Definition 2.1 states that the number of causes for any event should not be infinite. The second condition, known as conflict inheritance property, states that if an event e is in conflict with some event e', then it is in conflict with all causal successors of e'.

The concurrence relation co can be derived as follow:

$$co \stackrel{def}{=} E \times E \ - \ (\preceq \cup \succeq \cup \sharp),$$

where $\succeq$ denotes the reverse of causality relation.

It is useful to define one more auxiliary relation. Then, the immediate conflict relation $\sharp_m$ is defined as follows:$e \sharp_m e' \stackrel{def}{=} e \sharp e'$ and $\forall e_1, e_1' \in E : (e_1 \preceq e) \wedge (e_1' \preceq e') \wedge (e_1 \sharp e_1') \rightarrow (e_1 = e) \wedge (e_1' = e')$ [5]. The definition can be used to get the conflict relation of a quotient structure.

Clearly, two event structures $\mathcal{E}_1 = (E_1, \sharp_1, \preceq_1, l_1)$, $\mathcal{E}_2 = (E_2, \sharp_2, \preceq_2, l_2)$ are *isomorphic*, denoted $\mathcal{E}_1 \cong \mathcal{E}_2$, iff there exists a bijection: $f : E_1 \rightarrow E_2$ between E_1 and E_2 and preserving their action labels, casuality and confliction relation.

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

A configuration is a set of events that have happened during a specific run of the event structure. Conceptually a configuration can also be viewed as a global state.

Definition 2.2 (*Configuration*) Let $\mathcal{E}$ be an event structure.

- $X \subseteq E$ is left-closed, iff $\forall e, e' \in E$: $e \in X \wedge e' \preceq e \Rightarrow e' \in X$.
  $X \subseteq E$ is conflict-free iff $\forall e, e' \in X : \neg(e \sharp e')$.
- $X \subseteq E$ is a configuration of $\mathcal{E}$, iff X is conflict-free and left-closed. Let $Conf(\mathcal{E})$ denotes the set of all configuration of $\mathcal{E}$.

A configuration should be conflict free since conflicting events can never occur together in the same computation. For any event $e \in E$, $\downarrow e$ is defined as $\{e' \mid e' \preceq e\}$. In addition, all causal events of e in a configuration must be contained in the configuration.

3. Action-based event structure logic

3.1. Syntax

Let $Act = \{a,\ b, ...\}$ denote the set of actions. We fix $\mathcal{P} = \{p, q, ...\}$, a countably infinite set of atomic state propositions, and we also fix a set of atomic action propositions $\mathcal{A} = \{\widehat{a} \mid a \in Act\}$. We assume that $\mathcal{A}$ is disjoint with $\mathcal{P}$, and set $\mathcal{P}_A = \mathcal{P} \cup \mathcal{A}$, $\mathcal{P}_A$ is the set of atomic propositions. We let p, q with or without subscripts range over $\mathcal{P}$, and let a, b with or without subscripts range over $\mathcal{A}$. The set $\mathcal{F}_{aESL}$ of formulas of action-based event structure logic (for simple, aESL(C)) can now be built up inductively.

Definition 3.1*(Formula)*

F1 Every action proposition $\widehat{a} \in \mathcal{A}$ is a formula.

F2 Every state proposition $p \in \mathcal{P}$ is a formula.

F3 If α and β are formulas, them so are $\neg\alpha$ and $\alpha \wedge \beta$.

F4 If α is a formula, them so are $\Box\alpha$, $\overline{\Box}\alpha$.

F5 If α is a formula, them so are $\Box^\sharp\alpha$, $\Box^c\alpha$.

The last four clauses are the same as definitions of $ESL[C]$. The basic difference is that the atomic propositions in our logic consists of two parts, the one has no difference with the set of atomic propositions of $ESL[C]$, another is a new one which describe the property of actions.

The definitions of logical connectives and modalities such as $\vee$, $\leftrightarrow$, and $\diamond$ etc., is also the same as those of $ESL[C]$.

For example,

$\diamond\alpha \stackrel{def}{=} \neg\Box\neg\alpha$,

$\overline{\diamond}\alpha \stackrel{def}{=} \neg\Box\neg\alpha$,

$\diamond^\sharp\alpha \stackrel{def}{=} \neg\Box^\sharp\neg\alpha$,

$\diamond^c\alpha \stackrel{def}{=} \neg\Box^c\neg\alpha$.

We can extend the definition of atomic action formula to a subset of Act.

Definition 3.2 Let $A \subseteq Act$, then action formula $\widehat{A}$ of A is defined as follows:

$$\widehat{A} \stackrel{def}{=} \bigvee_{\forall a, a \in A} \widehat{a}.$$

3.2. Semantics

We start with the definition of frames.

Definition 3.3 A frame of $Fr = (E, \sharp, \preceq, l)$ is an arbitrary event structure.

Note that in some literatures (ES, LC_{ES}) where ES is an event structure and LC_{ES} is a configuration structure, is used as frame instead of an arbitrary event structure, but as for prime event structure, this does not change the idea.

Definition 3.4 A model is an ordered pair $M = (Fr, V)$, where

- $Fr = (E, \sharp, \preceq, l)$ is a frame,
- $V : E \longrightarrow 2^{\mathcal{P}}$ is a valuation function.

Let $M = (Fr, V)$ be a model with $Fr = (E, \sharp, \preceq, l)$. Let $p \in \mathcal{P}$, $\widehat{a} \in \mathcal{A}$, and $\alpha, \beta \in \mathcal{F}_{aESL}$. Then the notion of a formula α being true at event $e \in E$ in the model M, written as $M, e \models \alpha$, is defined inductively as follows:

F1 $M, e \models \widehat{a}$, iff $l(e) = a$.

F2 $M, e \models p$ iff $p \in V(e)$.

F3 $M, e \models \neg\alpha$ iff $M, e \not\models \alpha$.
$M, e \models \alpha \wedge \beta$ iff $M, e \models \alpha$ and $M, e \models \beta$.

F4 $M, e \models \Box\ \alpha$ iff $\forall e' \in E$, $e \preceq e'$ implies $M, e' \models \alpha$.
$M, e \models \overline{\Box}\ \alpha$ iff $\forall e' \in E$, $e' \preceq e$ implies $M, e' \models \alpha$.

F5 $M, e \models \Box^\sharp\ \alpha$ iff $\forall e' \in E$, $e \sharp e'$ implies $M, e' \models \alpha$.
$M, e \models \Box^c\ \alpha$ iff $\forall e' \in E$, $e\ co\ e'$ implies $M, e' \models \alpha$.

We only explain the first clause, since the others are standard in $ESL[C]$ and need no explanations. The first clause states that an atomic action proposition $\widehat{a}$ is true at event e, only if the label of e is a.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Obviously, any two atomic action propositions are inconsistent, they cannot be satisfied at an event together.

We also wish to draw attention to the semantics of propositions such as $\widehat{A}$.

- $M, e \models \widehat{A}$, iff $(\exists a \in A)$, $M, e \models \widehat{a}$.

The meaning of this clause is that if $\widehat{A}$ is true at event e, then there exists exactly in A one action labeling e.

The notions of satisfiable and valid are standard and not discussed here.

3.3. The proof system for aESL[C]

In order to reason about action based behaviour, the axioms of $aESL[C]$ should contain not only the axioms inherited from $ESL[C]$, but also some new axioms which characterize property related to (atomic) action propositions. In this paper, we only present the new axioms and then provide some explanatory remarks.

Axioms:

A1 $\widehat{a} \rightarrow \neg\widehat{b}$

A2 $True \rightarrow \widehat{Act}$

Axiom A1 indicates that any two atomic action propositions are inconsistent, and axiom A2 states that for each consistent formula, at least one atomic action proposition is satisfiable. Now, we give some properties of action propositions.

Lemma 3.5 Suppose $a, b \in Act$, $A, B \subseteq Act$, then

1. $\neg\widehat{a} = \widehat{Act - \{a\}}$, $\neg\widehat{A} = \widehat{Act - A}$,
2. $\widehat{a} \wedge \widehat{b} = False$,
3. $\widehat{a} \wedge \widehat{A} = \widehat{A \cap \{a\}}$, $\widehat{A} \wedge \widehat{B} = \widehat{A \cap B}$,

Proof: The result follows by appealing to the definition and semantics of action propositions and axioms A1, A2. □

Theorem 3.6 (Soundness w.r.t. the models)
For any set L of formulas and for any formula α in $\mathcal{F}_{aESL}$, the following statements hold:

1. if $\vdash \alpha$, then $\models \alpha$.
2. if $L \vdash \alpha$, then $L \models \alpha$.
3. if L is satisfiable in a model. then L is consistent.

Proof: 1. We will just argue for the soundness of the new two axioms A1 and A2.

Let $M = (Fr, V)$ be a model with $Fr = (E, \sharp, \preceq, l)$. Let $e \in E$, $a, b \in \mathcal{A}$, and $\alpha \in \mathcal{F}_{EESL}$. First consider A1, suppose $\alpha = \widehat{a}$ and $M, e \models \widehat{a}$. According to the semantics of atomic action propositions, $l(e) = a$. Since each action name is unique, $a \neq b$, that is, $l(e) \neq b$, thus $M, e \models \neg\widehat{b}$. Therefore axiom A1 is soundness.

Next consider A2. A2 states that $\widehat{Act}$ is a tautology. Let e be an arbitrary event, $l(e) \in Act$. Without the loss of generality, let $l(e) = a$, then $\forall e \in E, \exists a \in Act : M, e \models \widehat{a}$. According to the definition for $\widehat{Act}$, it always holds, thus $\models \widehat{Act}$. Hence, axiom A2 is soundness.

Statements 2 and 3 follow immediately from 1. □

In the following part we will prove completeness of our axiom system.

Theorem 3.7 The axiom system for aESL[C] is complete.

Since the proof for completeness of aESL[C] is not the goal of this paper, we give only the conclusion and won't discuss it here. □

3.4. EESL[C]-Simplified form of aESL[C]

The formulas of aESL[C] contain not only action propositions, but also state propositions. This may bring confusion between states and action executions, and the relation between action propositions and state propositions shows no clear and runs beyond comprehension. Thus, we won't hope the occurrence of action propositions in the formulas. We modify the modalities, by replacing $\Box$, $\overline{\Box}$, $\Box^{\sharp}$, and $\Box^{c}$ with $\Box_A$, $\overline{\Box}_A$, $\Box^{\sharp}_A$, and $\Box^{c}_A$ respectively, where $A \subseteq Act$. We add a new modality $\mathbb{A}$ which represents the present state within the constraint of actions set A.

Definition 3.8 The definitions of $\mathbb{A}$, $\Box_A$, $\overline{\Box}_A$, $\Box^{\sharp}_A$, and $\Box^{c}_A$ are as follows:

- $\mathbb{A}\alpha \stackrel{def}{=} \widehat{A} \wedge \alpha$,
- $\Box_A\alpha \stackrel{def}{=} \Box(\mathbb{A}\alpha) = \Box(\widehat{A} \wedge \alpha)$,
- $\overline{\Box}_A\alpha \stackrel{def}{=} \overline{\Box}(\mathbb{A}\alpha) = \overline{\Box}(\widehat{A} \wedge \alpha)$,
- $\Box^{\sharp}_A\alpha \stackrel{def}{=} \Box^{\sharp}(\mathbb{A}\alpha) = \Box^{\sharp}(\widehat{A} \wedge \alpha)$,
- $\Box^{c}_A\alpha \stackrel{def}{=} \Box^{c}(\mathbb{A}\alpha) = \Box^{c}(\widehat{A} \wedge \alpha)$,

where α is a formula including no action propositions.

This type of expressions without explicit action propositions can be viewed as a new logic system. We call the new logic extended event structure logic (EESL[C]), and denote the set of formulas of EESL[C] as $\mathcal{F}_{EESL}$.

We discover, however, that the set $\mathcal{F}_{aESL}$ of formulas of aESL[C] is actually equivalent to $\mathcal{F}_{EESL}$. It is an interesting thing, we'll show it in the following paragraph.

Theorem 3.9 $\mathcal{F}_{EESL}$ = $\mathcal{F}_{aESL}$.

Proof: The proof of this theorem consists of two parts. First, we'll show that $\mathcal{F}_{EESL}$ is a subset of $\mathcal{F}_{aESL}$. According to Definition 3.8, it is obvious.

Then, the fact that $\mathcal{F}_{aESL}$ is a subset of $\mathcal{F}_{EESL}$ is proved. It can be obtained inductively by the construction of formulas of aESL[C].

If $\widehat{a}$ is an action atomic proposition, then $\widehat{a}$ can be expressed as $\widehat{\{a\}} \wedge True$, let $A = \{a\}$, then $\widehat{a} = \mathbb{A} \wedge True \in \mathcal{F}_{EESL}$.

For any $p \in \mathcal{P}$, it can be written as $\widehat{Act} \wedge p$, let $A = Act$, thus $p \in \mathcal{F}_{EESL}$.

If α, $\beta \in \mathcal{F}_{EESL}$, then $\alpha \wedge \beta \in \mathcal{F}_{EESL}$. Since α can be written in the form of $\widehat{A'} \wedge \alpha'$, and β as $\widehat{B'} \wedge \beta'$, thus $\alpha \wedge \beta = (\widehat{A'} \wedge \widehat{B'}) \wedge (\alpha' \wedge \beta')$. According the third statement of Lemma.1, there exists one set $A = A' \cap B'$, such that $\widehat{A} = \widehat{A'} \wedge \widehat{B'}$. So, formula $\alpha \wedge \beta$ belongs to $\mathcal{F}_{EESL}$.

In the same way, we can testify that if $\alpha \in \mathcal{F}_{EESL}$, then $\Box\alpha$, $\Box\alpha$, $\Box^{\sharp}\alpha$, and $\Box^{c}\alpha \in \mathcal{F}_{EESL}$. Thus, we get the result.

Then, $\mathcal{F}_{EESL}$ = $\mathcal{F}_{aESL}$. □

Since $aESL[C]$ and $EESL[C]$ represent the same logic, in the following text, we use $aESL[C]$ as the unique name of this logic.

Theorem 3.10 $\mathcal{F}_{ESL} \subseteq \mathcal{F}_{aESL}$

Here, $\mathcal{F}_{ESL}$ denotes the set of formulas of $ESL[C]$.

Proof: In fact, each formula of $ESL[C]$ either has a counterpart in the $aESL[C]$ or can be expressed by a formula by replacing the modalities with the corresponding modalities where the subscript $A = Act$. Thus, this lemma is obviously correct. □

So, $aESL[C]$ is strictly more expressible than $ESL[C]$.

3.5. Expressiveness of aESL[C]

Since ESL[C] is a subset of aESL[C], all the properties expressible in ESL are obviously expressible in aESL[C]. Such as, safety property can be expressed as $\Box\alpha$ which states that α holds at all events of an event structure. $\Box^{c}\alpha$ expresses properties about concurrency relation. Moreover, properties relating action to state can be specified. In this section we will introduce a new conception for event structures: safety property under a set A of actions.

Definition 3.11 Let $M = (\mathcal{E}, V)$ be a model where $\mathcal{E} = (E, \sharp, \preceq, l)$ is an event structure, Act is the set of actions and $\mathcal{P}$ is a set of atomic state propositions, $B \subseteq E$ and $p \in \mathcal{P}$,

1. $e \in E$ is p-safe iff $p \notin V(e)$,
2. B is p-safe iff $\forall e \in B$, e is p-safe,
3. Model M is p-safe iff $\forall e \in E$, e is p-safe.

Definition 3.12 Let $M = (\mathcal{E}, V)$ be a model where $\mathcal{E} = (E, \sharp, \preceq, l)$ is an event structure, Act is the set of actions and $\mathcal{P}$ is a set of atomic state propositions, $A, B \subseteq E$ and $p \in \mathcal{P}$,

1. $e \in E$ is p-safe under A iff $l(e) \in A$ implies $p \notin V(e)$,
2. B is p-safe under A iff $\forall e \in B$, e is p-safety under A,
3. Model M is p-safe under A iff $\forall e \in E$, e is p-safety under A.

Through the comparison between definitions of safety and safety under a set A of actions, we notice that the former is more strict than the latter. That is to say, for some safe restriction p, one model which is p-safe must be p-safe under any set of actions, the reverse, however, may not be correct. This characteristic is useful in the specification of multi-agent system. We will show an example in the following paragraph.

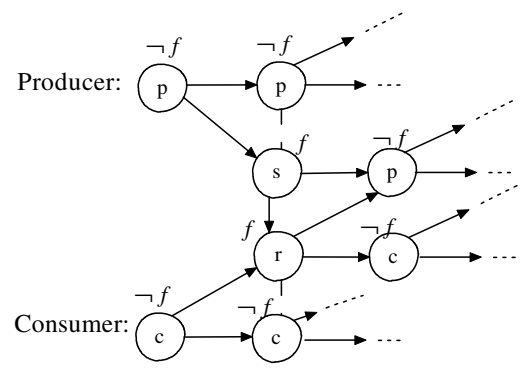


Figure 1. A simple event structure

Example.1 Consider an event structure in Fig 1. This structure contains two agents: producer and consumer. The producer continues providing data to a buffer. This behavior can be expressed as an action p. Once the buffer fulling of data, the producer sends a signal to consumer, this behavior denoted as action s. The consumer continues computing the data (action label: c) until getting a signal from producer. Then producer read data from the full buffer (action label: r).

Circles in the figure denote events. Text within the circle represents an action label of the event. Symbol outer of the circle is atomic state proposition: valuation of the event. Thus, $\{p, s\}$ is the set of producer's actions, while the set of actions of the consumer is $\{c, r\}$. The casual relation in figure 1 is expressed by arrow lines, and the conflict relation is depicted by dashed.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

We hope that the event structure possesses the property "producer won't provide data whenever the buffer is full", in other word, "the producer provide data only when the buffer is not full". We use proposition f express "the buffer is full". Thus, the property can be viewed as f-safety property under set $\{p\}$ and can be expressed by $\Box_A \neg f$, where $A = \{p\}$.

4. Symmetry reduction in event structures

Generally, *a permutation group* is defined on the basis of *a symmetric group*.

Definition 4.1 *(symmetric group)* Let X be a set, then $S_X := \{f : X \to X \mid f \text{ is a bijection}\}$ is a group with the compositions of maps as the binary operations. S_X is called a symmetric group on X.

Here, for any bijection $f \in S_X$, f is called a *permutation*.

Definition 4.2 *(permutation group)* Let S_X be the symmetric group on the set X, any subgroup of S_X is called a permutation group on X.

Now, we know that a permutation group over a set X consists of bijections: $X \to X$ and their compositions as the binary operations. Obviously, a symmetric group is a special permutation group. A permutation group over a set has good properties, in particular, it can induce an equivalence relation. This property is easily proved [19].

Definition 4.3 Let G be a permutation group on the set X and R be a relation on X. R is an equivalence relation, if there exists a permutation $f \in G$ such that $R = \{(a, b) \mid f(a) = b\}$.

The equivalence classes induced by an equivalence relation on a set form a partition of this set. Thus, given a set X, if there exists a permutation group on the set X, the permutation group can induce a partition of the set X. In this paper permutation groups are used to partition the set of events in an event structure so that we can use equivalence classes of events to investigate symmetry in the event structure.

4.1. Automorphism groups

Definition 4.4 *(auotmorphism)* Let $\mathcal{E} = (E, \sharp, \preceq, l)$ be an event structure and let G be a permutation group on the event set E. A permutation $f \in G$ is said to be an automorphism of $\mathcal{E}$ iff f satisfies the following condition: $\forall e_1, e_2 \in E$:

- $e_1 \preceq e_2 \Rightarrow f(e_1) \preceq f(e_2)$,
- $e_1 \sharp e_2 \Rightarrow f(e_1) \sharp f(e_2)$,
- $l(e_1) = l(f(e_1))$.

A permutation group G is called an *automorphism group* for the event structure $\mathcal{E}$ iff every permutation $f \in G$ is an automorphism of $\mathcal{E}$. Notice that because every $f \in G$ has an inverse, which is also an automorphism.

It is easy to see that if every permutation of the group G is an automorphism of $\mathcal{E}$, then the group G is an automorphism group for $\mathcal{E}$.

4.2. Quotient structure of an event structure

Let G be a permutation group acting on the event set E and let $e \in E$ and $\sim_G$ be the equivalence relation induced by G, then the equivalence class of an event $e \in E$ is defined as an events set $\theta(e) = \{e' \mid \exists f \in G : f(e) = e'\}$, which is called an *orbit*. Intuitively, the quotient structure $\mathcal{E}_G$ of an event structure $\mathcal{E}$ is obtained by collapsing all the events in one orbit to a single event of quotient structure. Next, we will give the formal definition of quotient structure of an event structure.

Definition 4.5 *(quotient structure of an event structure)* Let $\mathcal{E} = (E, \sharp, \preceq, l)$ be an event structure and let G be an automorphism group on the set E. The *quotient structure* $\mathcal{E}_G = (E_G,\ \sharp_M,\ \preceq_G,\ l_G)$ of $\mathcal{E}$ is defined as follows:

- $E_G = \{\theta(e) \mid e \in E\}$, the set of orbits of the events in E,
- $\sharp_G = \{(\theta(e_1), \theta(e_2)) \mid (e_1, e_2) \in \sharp_m\}$, $\sharp_m$ is the immediate conflict relation of $\mathcal{E}$,
- $\preceq_G = \{(\theta(e_1), \theta(e_2)) \mid (e_1, e_2) \in \preceq\}$,
- $l_G(\theta(e)) = l(e)$.

Note that the relation $\sharp_G$ does not represent the conflict relation, however, it is just the sign that the events in "conflict" can not occur adjacently. We can also define in quotient structure another conflict relation $\sharp'_G$, for example, let $\sharp'_G = \{(\theta(e_1), \theta(e_2)) \mid (e_1, e_2) \in \sharp\}$. In fact, the definition of $\sharp'_G$ says that two events have the relation $\sharp'_G$ can not be executed concurrently, which does little help to model reduction.

We hope that the quotient structure preserves all the properties of the original event structure. Unfortunately, the quotient structure $\mathcal{E}_G$ does not need to be an event structure since the conflict inheritance property may not preserve. However, the quotient event structure $\mathcal{E}_G$ provides some useful information. It is easy to check that $\mathcal{E}_G$ preserves all the properties of $\mathcal{E}$ except possibly the conflict relation. Instead, $\mathcal{E}_G$ satisfies another important condition which preserves the immediately confliction relation and

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

makes it possible to prove $\mathcal{E}_G$ to be modally equivalent to $\mathcal{E}$. The modally equivalence can be used to verify the property with related to conflict relation. We won't consider the question in this paper.

5. Computer-based systems and their event structures

We consider only K-sequential agents systems. That is to say, agents of a computer-based system can behave concurrently, however, each of them is sequential. First, the definition of K-sequential agents systems is given.

5.1. K-sequential agents systems

Definition 5.1[20] A K-sequential agents system is a structure P with the following components:

- K is a finite set of agents (or components),
- for each agent $i \in K$, a finite non-empty set S_i, called a set of local state of the agent i,
- for each agent $i \in K$, $s_i^0 \in S_i$ is the initial state of the agent i,
- for each nonempty agent set $X \subseteq K$, $i \in K$, $\Gamma_X \subseteq (\prod_{i \in X} S_i)^2$ is transition relation.

The transition-relation Γ_X models the events in which all agents in X participate. For each transition $t = (s, s_1)$, proc(t) denotes the set of agents involved in executing t. Let $\Sigma = \bigcup_{X \subseteq K} \Gamma_X$, The independency relation $I \subseteq \Sigma \times \Sigma$ of transitions is defined as follows: $(t, t') \in I$ if $proc(t) \cap proc(t') = \emptyset$. Thus, (Σ, I) is an independence alphabet. The dependency relation $D = \Sigma \times \Sigma - (I \cup id_\Sigma)$. The immediate dependency relation $D_m = \{(t, t') \in D \mid in(t)|_i = in(t')|_i$, for some $i \in proc(t) \cap proc(t')\}$, where $s|_i$ denotes the projection of s to the i-component.

Let $GS = \prod_{i \in K} S_i$ be the set of global states of P. A transition $t = (s, s') \in \Gamma_X$ is said to be enabled from a global state $g \in GS$, denoted as $t \in enabled(g)$, if $g|_X = s$. T denotes the set of all traces over (Σ, I).

Given two global states $g, g' \in GS$, $g \xrightarrow{t} g'$ iff for some $Y \subseteq K$, $t = (g|_Y, g'|_Y) \in \Gamma_Y$ and $g|_{K-Y} = g'|_{K-Y}$. An execution sequence $w = t_0 ... t_n \in \Sigma^*$ of P is a finite sequence of transitions such that there is a sequence of global states $\xi = g_0 g_1 ... g_n$ of P with $g_0 = (s_1^0 ... s_K^0)$, and $g_i \xrightarrow{t_i} g_{i+1}$, for each $i < n$.

Now, consider the traces of a K-sequential system P. First, define $\equiv$ as the least congruence in the string monoid $(\Sigma^*, \circ, \varepsilon)$ where $\circ$ denotes the concatenation operation such that $\forall a, b \in \Sigma : (a, b) \in I \Rightarrow ab \equiv ba$. Then, a trace is an equivalence class induced by $\equiv$. The trace generated by a transition t is denoted as $[t]$. Let $[\Sigma^*] = \{[w] \mid w \in \Sigma\}$ be the set of all traces oven (Σ, I). The successor relation $\rightarrow$ is defined as follows: $[w_1] \rightarrow [w_2]$ iff there is a transition $t \in \Sigma$ such that $[w_1][t] = [w_2]$. The prefix relation $\preceq$ in $[\Sigma^*]$ is the transitive and reflexive closure of $\rightarrow$. Obviously, $([\Sigma^*], \preceq)$ is a partial order set.

We say that two traces $[w'], [w'']$ are consistent if there is a trace $[w] \in [\Sigma^*]$ such that $[w'] \in\downarrow_{\preceq} [w] \wedge [w''] \in\downarrow_{\preceq} [w]$. Otherwise, they are in conflict. For each $w \in \Sigma^*$, let $last(w) = a$ if $w = w'a$ for some $w' \in \Sigma^*$. For each trace $\tau \in [\Sigma^*]$, $Max(\tau) = \{last(w) \mid w' \in \tau\}$.

A traces τ is prime if $|Max(\tau)| = 1$. For the system P, prime execution sequences of P are those execution sequences which are prime, denoted as T.

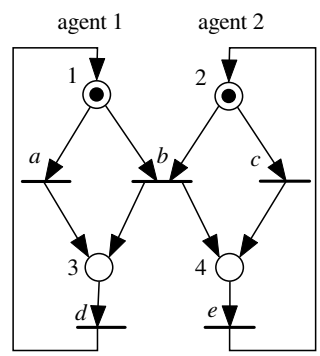


Figure 2. 2-agents system CC

Example.2 A 2-agents system CC is show in Fig.2. It is composed of two agents which carry through a collaborating computation task. Each agent gets data from either own computation or from another agent. The local states of agents are denoted with circles, whereas the transitions with bars. In this example, the set of transitions $\Sigma = \{a, b, c, d, e\}$, $S_1 = \{1, 3\}$, $S_2 = \{2, 4\}$ and $d = ((1, 2), (3, 4))$. $[ad], [bd], [ce]$ are the prime traces of CC, while $[ac]$ and $[bde]$ are not, since $Max([ac])$ = $\{a, c\}$ and $Max([bde]) = \{d, e\}$.

5.2. The event structures for computer-based systems

Let P be a multi-agents system, the corresponding event structure of P can be easily obtained from the prime execution sequences.

Definition 5.2*(Event structures for K-sequential agents systems)*
Let T be the set of prime execution sequences of P and $\preceq_T$ be the prefix relation of T. An event structure $\mathcal{E}_P = (E_P, \sharp_P, \preceq_P, l_P)$ for the system P is defined as follows:

- $E_P = T$, is the set of events,

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

- $[t_1] \sharp_P [t_2]$, if there exist $t_1' \in [t_1]$ and $t_2' \in [t_2]$ are in conflict,
- $\preceq_P = \preceq_T$, casuality relation,
- $l_P(e) = Max(e)$, for $e \in E_P$.

According to the definition, we can get an event structure for system CC of Example.2.

Example.3 The event structure for system CC is shown in Fig.3. Each event is denoted with an ellipse. The representations of casuality relation and conflict relation are same to those in Example.1. The last bold action name of each trace in the bracket represents the label of corresponding event, *eg.*, $l([bde\mathbf{b}]) = b$.

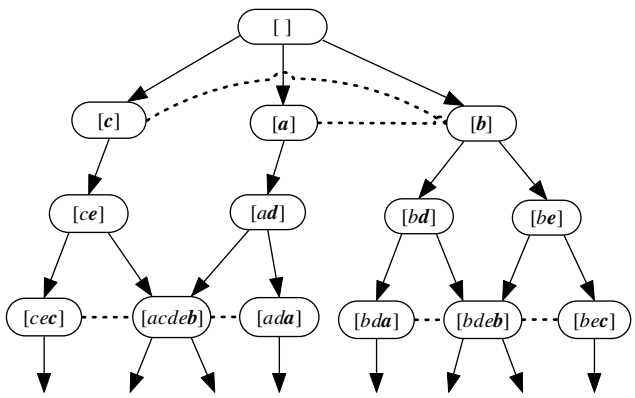


Figure 3. The event structure for CC

Obviously, according to the definition of permutation of an event structure, we can get a symmetry group G for CC. The following gives two permutations of G:

$$f_1 \left(\begin{array}{ccccccccc} [\,] & [a] & [c] & [b] & [ad] & [ce] & [bd] & [be] & [bda] \\ [\,] & [ada] & [cec] & [acdeb] & [adad] & [cece] & [bdebd] & [bedbe] & [bdada] \end{array} \begin{array}{cccc} [bec] & [bdeb] & \dots \\ [becec] & [bdebedb] & \dots \end{array} \right)$$

$$f_2 \left(\begin{array}{ccccc} [\,] & [a] & [c] & [b] & [ad] \\ [\,] & [adada] & [cecec] & [acdeacdeb] & [adadad] \end{array} \begin{array}{ccccc} [ce] & [bd] & [be] & [bda] & \dots \\ [cececе] & [bdebdebd] & [bedbedbe] & [bdadada] & \dots \end{array} \right)$$

The permutations f_1, f_2 are clearly automorphism. So, the group G is an automorphism group. Then, the orbit of trace $[ad]$ is the form $[(ad)(ad)^*]$.

Thus, we can get the quotient structure of the event structure for system CC.

Example.4 The quotient structure of event structure for CC is shown in Fig.4. This figure shows that the conflict relation inherited from original event structure does not preserve the confliction property. In the quotient structure, conflict relation only denotes that the events in conflict can not executed adjacently and concurrently.

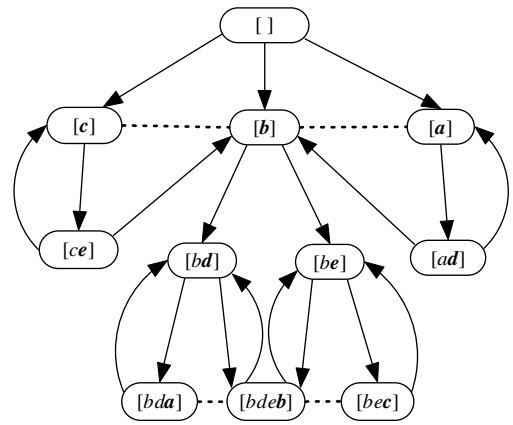


Figure 4. The quotient structure of event structure for CC

In the next section, we will discuss the model checking of $aESL[C]$ over the quotient structure of event structure for computer-based systems.

6. Model checking of $aESL[C]$

The model checking problem of $aESL[C]$ can be described as follows: given a model $M = (\mathcal{E}_P, V)$ where $\mathcal{E}_P = (E_P, \sharp_P, \preceq_P, l_P)$ is a quotient structure of event structure that represents a computer-based system and an $aESL[C]$ formula α expressing some desired specification, determine whether $M \models \alpha$.

Theorem 6.1 There is a deterministic algorithm for determining whether an $aESL[C]$ formula α (without conflict modality) is satisfied in a model $M = (\mathcal{E}_P, V)$ where $\mathcal{E}_P = (E_P, \sharp_P, \preceq_P, l_P)$ is a quotient structure of event structure.

Proof: Assume that formula α is one of the following forms: atomic proposition, $\neg\alpha_1$, $\alpha_1 \vee \alpha_2$, $\diamond\alpha_1$, $\overline{\diamond}\alpha_1$, $\diamond^c\alpha_1$. In fact, each formula (except for conflict modality) can be transformed into one of the above six cases. We only discuss the form $\diamond\alpha_1$ and $\diamond^c\alpha_1$, since $\overline{\diamond}\alpha_1$ is the reverse of $\diamond\alpha_1$ and the other cases are straightforward. Consider formula $\diamond\alpha_1$, model checking algorithm first finds all events which are labeled by α_1. Then, it goes along with the relation $\succeq$ and label all the casual predecessor with $\diamond\alpha_1$. The case for $\diamond^c\alpha_1$ is a little complex. The algorithm establishes three set S, P and C for each event e labeled with α_1

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

to represent the events that can not executed concurrently with e. It first goes forward using the casuality relation and add all the events which can be reached by a path to set S, then the algorithm add all the predecessor of e into another set P, and add all events that is in conflict with some events in P to set C. Then all the events outer of the three set are labeled with $\diamond^c\alpha_1$. This step requires time $O((|E_P| + | \preceq_P |)^2(|E_P| + |\sharp_P|))$.

Since the algorithm operates in stages. The i-th stage handles all sub-formulas of α of length i ($i \leq |\alpha|$). Thus, the algorithm requires time $O(|\alpha|(|E_P| + | \preceq_P |)^2(|E_P| + |\sharp_P|))$. So, the model checking algorithm for $aESL[C]$ formulas without conflict modality against quotient structures of event structures for K-sequential agent systems is decidable. □

7. Conclusion

We have introduced an action-based event structure logic $aESL[C]$ which in fact is the same as the extended event structure logic with action propositions and given some characteristic of it. $aESL[C]$ is an expressive logic to specify the behavior of multi-agent systems. We define a quotient structure of event structure for multi-agent systems. This is a key step to obtain a model checking algorithm for computer-based systems. We show that algorithm for model checking the quotient structure for $aESL[C]$ formulas is decidable.

The algorithm does not consider the conflict formulas. We hope to complete it in the the future work.

References

[1] G. Boudol. Flow event structures and flow nets. In *Semantics of Systems of Concurrent Processes, Proceedings LITP Spring School on Theoretical Computer Science*, volume 469 of *Lecture Notes in Computer Science*, pages 62–95. Springer-Verlag, 1990.

[2] E. M. Clarke and E. A. Emerson. Design and synthesis of synchronization skeletons using branching-time temporal logic. In *Pro. of the Workshop on Logic of Programs*, volume 131 of *Lecture Notes in Computer Science*, pages 52–71. Springer-Verlag, 1981.

[3] J. Jiang, J. Wu, and D. Chen. Symmetry in event structures. In *Proceedings of the 2006 ACM symposium on Applied computing*, pages 1850–1851. ACM Press, 2006.

[4] F. Laroussinie, S. Pinchinat, and P. Schnoebelen. Translation results for modal logics of reactive systems. In *Proc. of AMAST'93*, pages 297–306. Springer, 1993.

[5] K. Lodaya, M. Mukund, R. Ramanujam, and P. S. Thiagarajan. Models and logics for true concurrency. Technical report, Ins. of Mathematical Sciences, Internal Report TCS-90-3, Madras, India, 1990.

[6] K. Lodaya and P. S. Thiagarajan. A modal logic for a subclass of event structures. volume 267 of *Lecture Notes in Computer Science*, pages 290–303. Springer-Verlag, 1987.

[7] P. Madhusudan. Model-checking trace event structures. In *Proceedings of the eightteenth Annual IEEE Syposium on Logic in Computer Science*, pages 371–380, Los Alamitos, CA, 2003. IEEE Computer Society.

[8] M. Mukund and P. S. Thiagarajan. Logical characterization of well branching event structures. *Theoretical Computer Science*, 96:35–72.

[9] M. Mukund and P. S. Thiagarajan. An axiomatization of event structures. volume 405 of *Lecture Notes in Computer Science*, pages 143–160. Springer-Verlag, 1989.

[10] M. Müller-Olm, D. Schmidt, and B. Steffen. Model-checking: A tutorial introduction. In *Static Analysis*, volume 1694 of *Lecture Notes in Computer Science*, pages 330–354. Springer, 1999.

[11] R. Nicola and F. W. Vaandrager. Action versus state based logics for transition systems. In *Semantics of Systems of Concurrent Processes, Proceedings LITP Spring School on Theoretical Computer Science*, volume 469 of *Lecture Notes in Computer Science*, pages 407–419. Springer-Verlag, 1990.

[12] M. Nielsen, G. D. Plotkin, and G. Winskel. Petri nets, event structures and domains, part I. *Theoretical Computer Science*, 13(1):85–108, 1981.

[13] W. Penczek. A temporal logic for the local specification of concurrent systems. In *Proc. of the IFIP 11th World Computer Congress*.

[14] W. Penczek. Temporal logic of causal knowledge. In *Proc. of WoLLic'98*.

[15] W. Penczek. A temporal logic for event structures. *Fundamenta Informaticae*, 11(3):297–326, 1988.

[16] W. Penczek. Branching time and partial order in temporal logics. In *Time and logic: a computational approach*, pages 179–228. UCL Press Ltd., London, UK, 1995.

[17] S. Pinchinat, F. Laroussinie, and P. Schnoebelen. Logical characterizations of truly concurrent bisimulation. Technical Report 114, Grenoble, 1994.

[18] V. R. Pratt. Modeling concurrency with partial orders. *Int. J. of Parallel Programming*, 15(1):33–71, 1986.

[19] J. J. Rotman. *An Introduction to the Theory of Groups*. Springer, New York, USA, 1994.

[20] W. Penczek. Model checking for a subclass of event structures. In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 1217 of *Lecture Notes in Computer Science*, pages 145–164. Springer, 1997.

Aspect-Oriented Modeling of Ubiquitous Web Applications: The *aspectWebML* Approach

A. Schauerhuber*
M. Wimmer
Institute of Software Technologies and Interactive Systems, Vienna University of Technology, Austria
{as,mw}@isis.tuwien.ac.at

W. Schwinger
Department of Telecooperation University of Linz, Austria
wieland@schwinger.at

E. Kapsammer
W. Retschitzegger
Information Systems Group University of Linz, Austria
{ek,werner}@ifs.uni-linz.ac.at

Abstract

Ubiquitous web applications (UWA) are required to be customizable, meaning their services need to be adaptable towards the context of use, e.g., user, location, time, and device. Considering UWA's from a software engineering point of view, a systematic development on basis of models is crucial. Current web modeling languages, however, often disregard the crosscutting nature of customization potentially affecting all parts of a web application, and often mingle core and customization functionality. This leads to inefficient development processes, high maintenance overheads, and a low potential for reuse.

We regard customization as a crosscutting concern in the sense of the aspect-oriented paradigm. As a proof of concept, we extend the prominent web modeling language WebML on basis of our reference architecture for aspect-oriented modeling. This allows for a clear separation between the core and customization functionality, and – as a spin-off – demonstrates how to bridge existing (domain-specific) modeling languages with aspect-oriented concepts.

1. Introduction

With the emergence of mobile devices as new access channels to the Internet, we are now facing a new generation of web applications, called ubiquitous web applications. UWAs are characterized by the anytime/anywhere/anymedia paradigm, taking into account that services are not exclusively accessed through traditional desktop PCs but through mobile devices with different capabilities, by users with various interests at anytime from anyplace around the globe. Services provided by UWAs are adapted to the actual context of use in order to preserve or even enhance their semantic value for users. Thus, knowing the context, e.g., user, location, time, and device, and providing *adaptation operations* for web pages and their different kinds of contents, e.g., text, images, and links, are the main prerequisites for *customization* of web applications towards ubiquity. Customization then denotes the *mapping* of the required *adaptation* of an application's services with respect to its *context* [6].

Considering UWA's from a software engineering point of view, a systematic development on basis of models is crucial. There are already some approaches dealing with the ubiquitous nature of web applications and the model-driven development thereof, the most prominent examples being WebML [3], UWE [7], and OO-H [5] (for an overview methods and tools for web application development we refer to [15]). Concerning customization modeling, however, they are still in their early stages due to the following reasons. First, the provided customization mechanisms frequently do not allow to deal with all different parts of a web application in terms of its content, hypertext and presentation levels and their structural and behavioral features (cf. Figure 1), thus, disregarding the crosscutting nature of customization. Second, customization is often tangled with the core web application, thus, neither a *context model* nor *adaptation operations* enter web application models in

* Andrea Schauerhuber is PhD student at the Women's Postgraduate College for Internet Technologies and has been funded by the Austrian Federal Ministry for Education, Science, and Culture, and the European Social Fund (ESF) under grant 31.963/46-VII/9/2002.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

an explicit, self-contained and extensible way. This leads to inefficient development processes, high maintenance overheads and a low potential for reuse.

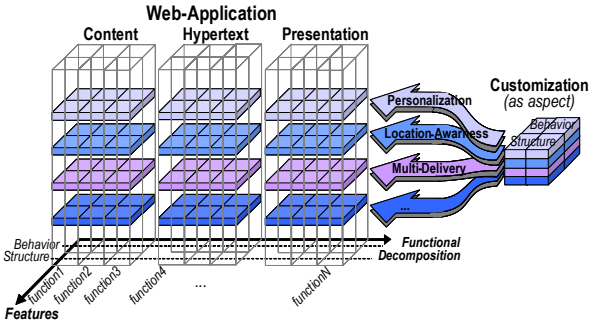


Figure 1. Customization as an aspect

To cope with these problems, we propose *aspectWebML* using aspect-orientation [12] as driving paradigm to incorporate customization in ubiquitous web applications at the modeling level (cf. Figure 1). As a proof of concept, we use our reference architecture for aspect-oriented modeling [13], which describes the necessary concepts of aspect-oriented modeling (AOM), as a blueprint for extending the MOF-based [9] metamodel of WebML [14], a prominent domain-specific language for modeling data-intensive web applications.

The benefits of this approach are fourfold. First, it takes into account the crosscutting nature of customization, allowing to influence all parts of a web application. Second, despite this omnipresence, a clear separation between the core services and customization functionality can be maintained. The core services of the web application remain oblivious to the need for customization, allowing even to make existing, non-ubiquitous web applications context-aware. Third, while our motivation for extending WebML has been driven by the need to separately capture customization, the extensions made also allow modeling of other aspects than the customization aspect. Finally, as a spin-off, it demonstrates how to bridge existing (domain-specific) modeling languages with aspect-oriented concepts.

The remainder of this paper is organized as follows. In Sec. 2 we outline our contributions with respect to related work and briefly introduce the WebML language using as a running example a *Museum* web application in Sec. 3. In Sec. 4, we report on how to bridge WebML to AOM according to the AOM reference architecture and present the specific AOM extensions to the WebML metamodel in terms of *aspectWebML*. In Sec. 5, we compare the original modeling approach of WebML with *aspectWebML* by extending a *Museum* web application with customization functionality and report on our prototype modeling editor. Finally, we conclude with an outlook on future work in Sec. 6.

2. Related Work

Currently, the majority of AOM approaches is first, based on UML and second, designed as general-purpose languages with respect to the application domain. We currently know of three UML-based approaches specific to a certain domain. In [4] and [11] two UML profiles have been proposed, the first one for modeling the notification aspect in CORBA applications and the second one for AOM in the web service domain. A third approach applies AOM in the domain of web application modeling [1], [17]. More precisely, while in [17], the UML-based web modeling language UWE has been extended with aspect-oriented concepts to model the *access control aspect* in web applications, the approach presented in [1] is closely related to our work in that it identifies *adaptivity* as a crosscutting concern in web applications. In particular, an extension of UWE's metamodel with aspect-oriented modeling techniques has been proposed and allows making navigation in web applications adaptive. Our approach, however, differs in three ways. First, we are building on a lean MOF-based metamodel of WebML, which has been established during our previous work [14], thus avoiding the unnecessary overhead of the huge UML metamodel. Second, modeling customization in UWE [1] currently is limited to the hypertext level of web applications and does neither support the content level nor the presentation level. Third, the aspect-oriented extensions applied to UWE are tailored to a specific aspect, only, being the access control aspect [17] and the navigation adaptivity aspect [1], respectively. In contrast to that, our approach is to use the AOM reference architecture as a blueprint to extend the WebML metamodel with AOM concepts, thus, allowing to model different aspects with one coherent set of concepts.

3. A WebML Primer

WebML is one of the most prominent modeling languages in the web modeling field due to existing tool support including a model editor, a code generation facility, and a runtime environment in form of the commercial WebRatio tool (www.webratio.org) and applications in real world projects. Following, we give a brief introduction into its modeling concepts using a *Museum* web application as a running example. The *Museum* web application is based on [2] and will be extended with customization functionality in Sec. 5.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

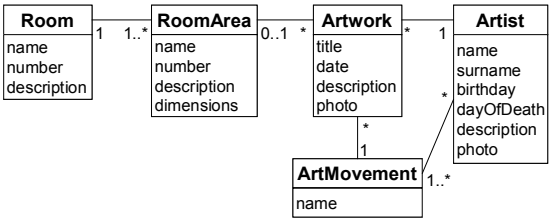


Figure 2. Museum content model

The content level of the *Museum* web application is represented by the *content model*, which – in WebML – is based on the Entity-Relationship model (cf. Figure 2, where we use the UML notation for multiplicities for readability purposes). The museum possesses a collection of *Artworks*, some of them being exhibited in certain *RoomAreas* of one of the museum's *Rooms*. A specific piece of *Artwork* belongs to a certain *ArtMovement* and has been created by a certain *Artist*.

The *hypertext model* of the *Museum* web application is based on the content model. Figure 3 shows eight web *Pages*, the majority of them containing so called *ContentUnits*, which allow to query the content model and to display the result on the Page (Please note, that the clouds in the Figure 3 represent comments and are not part of the hypertext model.).

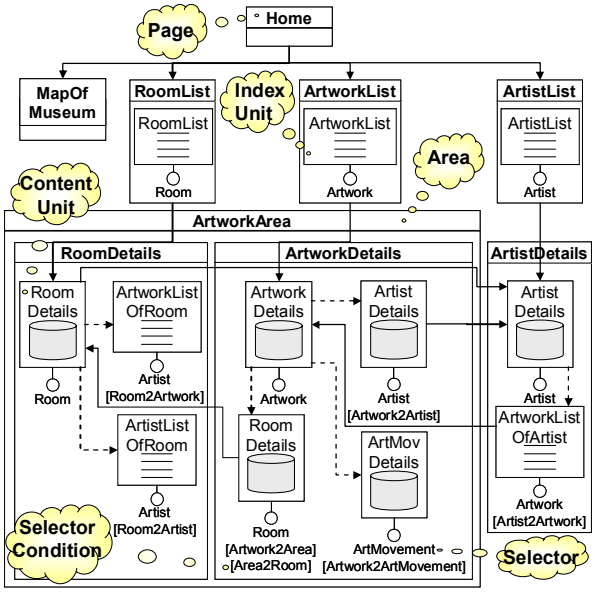


Figure 3. Museum hypertext model

The *Home* Page links four Pages. The *RoomList*, *ArtworkList*, and *ArtistList* Pages each contain one ContentUnit, a so called *IndexUnit*, which presents multiple instances of an entity type from the content model as a list. From these IndexUnits, a user then can navigate to the *RoomDetails*, *ArtworkDetails*, and *ArtistDetails* Pages, presenting further information according to a single instance of a *Room*, an *Artwork*, or an *Artist*. For example, the *RoomDetails* Page contains information about the *Room* itself, which is derived from the content model using a so called *DataUnit* named *Room*, i.e., a ContentUnit. ContentUnits select the information from the content model using a *Selector*, e.g., *Room* for DataUnit *Room*, and optionally several *SelectorConditions* depicted in square brackets. Additionally, the Page contains two IndexUnits listing *Artists* and *Artworks* exhibited in the specific *Room*. WebML also provides the container concept *Area*, such as *ArtworkArea,* which allows grouping Pages that deal with some related topic [3].

4. Bridging WebML to Aspect-Oriented Modeling

In this work, we make a step towards bridging WebML to AOM using as a basis our AOM reference architecture. Subsequently, we briefly introduce the WebML metamodel in Sec. 4.1 and our AOM reference architecture in Sec. 4.2. In Sec. 4.3, we provide detailed information on how we applied the AOM reference architecture to the WebML language.

4.1. The WebML Metamodel

A prerequisite for bridging WebML to AOM is the existence of a proper metamodel of the web modeling language, which allows to seamlessly hook up the aspect-oriented concepts. Similar to most web modeling languages, WebML – originally focusing on notational aspects – has been designed without using expressive object-oriented meta-modeling techniques, employing DTD's, only. To further complicate things, recent WebML language concepts – most notably its customization mechanisms [2] – have not been introduced into the WebML DTD but rather hard-coded directly within the WebML modeling tool. To cope with these problems, in previous work [14], we semi-automatically constructed a MOF-based metamodel draft for WebML on basis of the WebML DTD. For our purpose of modeling UWA's, we manually extended this metamodel by introducing also WebML's concepts for customization (cf. Sec. 5.1).

4.2. The AOM Reference Architecture

Our primary goal in designing the AOM reference architecture [13] was to establish a common understanding in the field of AOM. The reference architecture has been defined in terms of a UML class diagram and identifies the basic ingredients of aspect-orientation, abstracted from specific modeling languages. In this respect, it captures the important

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

AOM concepts, their interrelationships and even more importantly their relationships to an arbitrary modeling language, e.g., a general-purpose modeling language such as UML or any other domain-specific modeling language such as WebML. The AOM reference architecture, however, does not represent a language specification in terms of a metamodel itself, but rather can be used as a blueprint for designing new AOM languages or for extending existing (domain-specific) modeling languages with concepts of the aspect-oriented paradigm.

The AOM reference architecture comprises four major building blocks, each subsuming related concepts (cf. Figure 4). In the following we point out the most important concepts and refer the interested reader to [13].

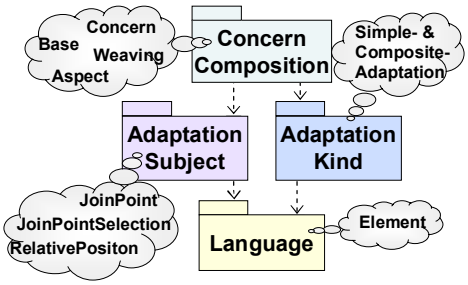


Figure 4. AOM reference architecture

The *ConcernComposition* package deals first, with the separation of a system's *Concerns* into appropriate units of modularization, i.e., *Base* and *Aspect*, and second, with their interrelationships, i.e., their composition by means of a *Weaving* specification. In the *AdaptationSubject*, we summarize concepts for identifying where to introduce an aspect's adaptation including *JoinPoint*, *JoinPointSelection*, and *RelativePosition* (denoting where to insert an aspect's adaptation relative to a join point, e.g., *before*, *after*, and *around*). The *AdaptationKind* package subsumes concepts to describe how an aspect adapts a concern, i.e. *Adaptation*. Finally, the *Language* package represents the language including its modeling *Elements* to be extended with aspect-oriented concepts.

4.3. The *aspectWebML* Metamodel

For designing *aspectWebML* we used our AOM reference architecture as a basis, meaning that its concepts and their interrelationships have not been adopted one-to-one. This is due to reasons concerning syntax on the one hand and reasons concerning design goals on the other hand. First, the AOM reference architecture has been defined in terms of a UML class diagram, while the WebML metamodel is MOF-based. Thus, we had to capture concepts available in UML, only, differently in the MOF-based *aspectWebML* metamodel. For example, we had to resolve association classes and replace aggregation associations with either composition associations or references. Second, in order to keep the language simple for the time being, we made some design decisions resulting in a more restrictive AOM language compared with our AOM reference architecture, e.g., we currently allow aspects to be woven into bases but not into aspects (cf. Figure 5).

4.3.1 The WebML Package. The AOM reference architecture assumes the modeling language to have a root element from which every modeling concept of the language inherits. This is necessary, since first, both *Base* and *Aspect* including its *Adaptations* are formalized by any set of modeling elements of the modeling language (cf. Figure 5: containment references from *Concern* to *ModelElement* and from *Aspect* to *Adaptation*), and second, *JoinPoints*, i.e., the locations where an aspect introduces its adaptations, are representations of elements of the modeling language. Since WebML originally did not provide such a root element, we reorganized the metamodel by introducing the abstract meta class *ModelElement*, having an attribute *isAdaptable* of type Boolean. This attribute – if set to *true* – allows to define the join point model of the AOM language, i.e. the meta classes of the modeling language that are allowed to serve as join points for aspects. While this represents an elegant solution, it required a change of WebML's metamodel. This could be avoided by simply duplicating all necessary references, e.g., from JoinPoint to the required modeling element of the language.

Currently, we are still investigating what kinds of adaptations in terms of aspect are meaningful within the realms of *aspectWebML*. Thus, we did not yet restrict the join point model to a subset of WebML's modeling concepts, meaning that every modeling concept can be subject of adaptations in *aspectWebML* models. This decision also reflects the ongoing discussion about join point models and adaptation effects in AOM.

4.3.2 The ConcernComposition Package. A model in *aspectWebML* consists of *Concerns*, which are either an instance of *Base* or of *Aspect*. An *Aspect* can be woven in to a *Base* by means of a *Weaving* specification. More specifically, the *Weaving* has *AdaptationRules*, which determine *where* (cf. Sec. 4.3.3) the *Aspect's Adaptations* have to be introduced in the *Base* and what kind of effect (cf. *AdaptationEffectKind* in Figure 5) these *Adaptations* imply.

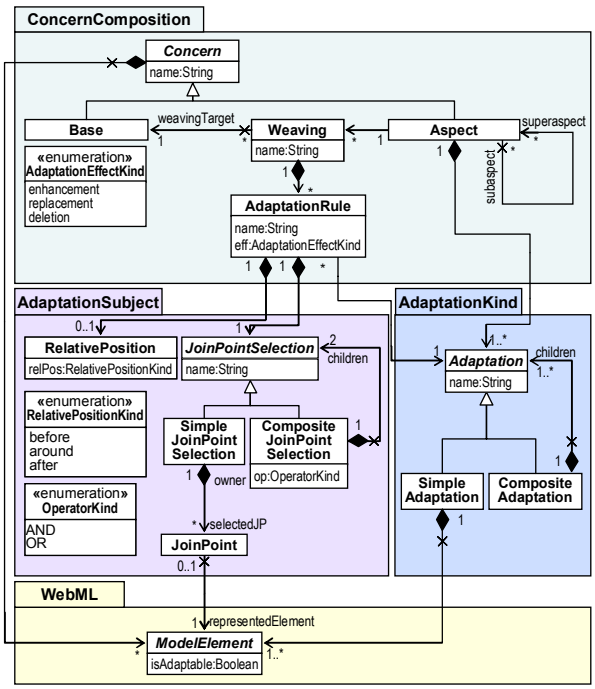


Figure 5. The *aspectWebML* metamodel

4.3.3. The AdaptationSubject Package. The adaptation hooks of a *Base* are represented by *JoinPoints*, which are identified by a *SimpleJoinPointSelection* (i.e., a pointcut). In addition, an *AdaptationRule* optionally may specify a *RelativePosition* where to insert *Adaptations* with respect to the selected join points. For reuse purposes, we allow *SimpleJoinPointSelections* to be composed to *CompositeJoinPointSelections* by means of *AND* and *OR* operators. Currently, our mechanism to select join points is limited to a manual identification of each single join point. Thus, for defining an instance of *SimpleJoinPointSelection* at modeling level, the user will instantiate join points from *JoinPoint* and link them to instances of *ModelElement*. The investigation of more elaborated join point selection mechanisms, such as OCL [10] or Join Point Designation Diagrams (JPDD) [15], and their applicability in *aspectWebML* is subject to future work.

4.3.4. The AdaptationKind Package. Adaptations consist of WebML *ModelElements*. For reuse purposes we distinguish between *SimpleAdaptations* and *CompositeAdaptations*, the latter allowing to combine existing *Adaptations* to form more complex ones.

5. Modeling Customization

In this section, we show how customization of the *Museum* web application (cf. Sec. 3) currently can be modeled with the original WebML language and point out the specific problems of the approach in Sec. 5.1. In Sec. 5.2, we present how to model the same application using *aspectWebML* and report on the prototype implementation of a model editor for *aspectWebML*.

5.1. Modeling Customization in WebML

In [2], WebML has recently been extended with concepts for modeling context-awareness, illustrated in a *Museum* web application example for which also a demo implementation has been provided (http://dblambs.elet.polimi.it/Demos/indexen.htm).

Following, we explain the necessary extensions to the original application (cf. Sec. 3) in order to model location-awareness, i.e., customization according to the location context. In particular, we want to model the following situation: If the visitor requests the *ArtworkDetails* Page, the specific *Artwork* of the *RoomArea* the visitor is currently in, shall be displayed. If, however, no *Artwork* is exhibited in the visitor's *RoomArea*, the visitor is redirected to the *RoomDetails* Page, which presents information about the room the visitor is currently in. In addition, the same set of adaptations shall be applied, if the visitor requests the *RoomDetails* Page.

It is assumed that an RFID-based location-sensing mechanism is available in the museum, that each visitor – or rather the mobile device s/he is using – has a unique RFID tag, and that the location-sensing infrastructure will continuously update the content model with the visitor's current location information.

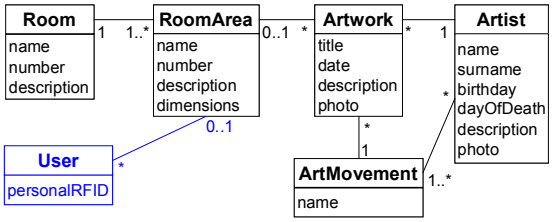


Figure 6. Location-aware content model

5.1.1 Customization in the Content Model. In WebML, the required context information is simply added to the ContentModel in terms of new Entity types, their Attributes, and their Relationships (cf. Figure 6). In the *Museum* web application, we need to know the user's location, i.e., the *RoomArea*. Thus, a *User* Entity type is introduced having a Relationship with *RoomArea*.

5.1.2 Customization in the Hypertext Model. In the HypertextModel, we use three of WebML's new concepts for modeling location-awareness: First,

ArtworkArea, *RoomDetails*, and *ArtworkDetails*, are marked as context-aware Areas and Pages, each having a so called *ContextUnit*. The semantics of *ContextUnits* is that they encapsulate context-aware behavior – also called *context clouds* – of Areas and Pages, which is executed before the actual Page computation, i.e., the computation of *ContentUnits*. When a context-aware Page is requested, then the context clouds of its containers from the outermost to the innermost are evaluated before the Page's context cloud. Second, *GetArea* and *GetArtwork*, so called *GetDataUnits*, allow querying the ContentModel, without displaying the content like other ContentUnits but providing it for further computation to the context cloud. Third, the *IFUnit* represents a control structure, which allows evaluating conditions and thus, may trigger different behavior in the context cloud.

Following, we describe the necessary additions to the HypertextModel of Figure 3 in order to model location-awareness (cf. Figure 7, where we omitted several parts of the original HypertextModel for readability reasons).

1. We add a ContextUnit to *ArtworkArea*, which now retrieves the users's location via *GetArea* every time either the *RoomDetails* or the *ArtworkDetails* Pages are requested. The *currentUser* represents a global parameter in the model, which can be retrieved by *GetUser*, a GetUnit.
2. We add a ContextUnit to *ArtworkDetails*, which retrieves the specific *Artwork* of the *RoomArea* the visitor is currently in, using the *GetArtwork* GetDataUnit. If, however, no *Artwork* is exhibited in the visitor's *RoomArea*, the visitor is redirected to the *RoomDetails* Page using an *IFUnit*.
3. We replace the default SelectorCondition (typically not shown in WebML models) of DataUnit *RoomDetails*, which always uses the ID for retrieving an Entity instance of *Room*, with a SelectorCondition *[RoomArea2Room]*, since the *RoomDetails* Page now has to present information about the visitor' current room.
4. The same set of adaptations shall be applied for the *RoomDetails* Page. Thus, we only need to add a ContextUnit to the *RoomDetails* Page which then links to the previously added *GetArtwork* GetDataUnit.

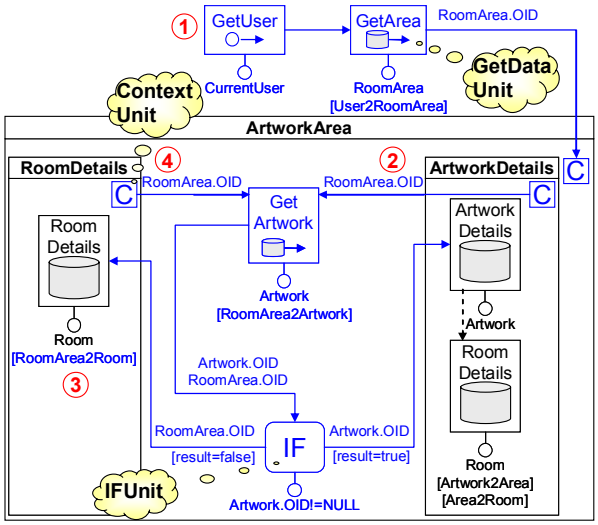


Figure 7. Location-aware hypertext model

5.1.3 Deficiencies of the WebML approach. Currently, if customization functionality is introduced to a web application model in WebML by enhancing, replacing, or deleting modeling elements, developers face the following problems: First the original web application model is lost. Second, it is not clear what modeling elements make up customization functionality. And third, customization functionality, that is scattered across WebML models and hampers their readability.

5.2. Modeling Customization in aspectWebML

Unlike WebML, *aspectWebML* allows introducing new functionality into all parts of a web application model but – at the same time – maintains a clear separation between the original model and the new functionality in terms of *Aspects* as is exemplified in Figure 8. For want of a concrete syntax for *aspectWebML*, we currently present *aspectWebML* models in terms of UML object models and trees, i.e., our model editor's view (cf. Sec. 5.2.3).

In Figure 8 (b), we present an overview of the *Museum* web application model defined in *aspectWebML*. This specific *aspectWebML* model consists of the *Museum Base*, i.e., the original *Museum* web application consisting of a *ContentModel*, a *HypertextModel*, and a *PresentationModel* (cf. Sec. 3), the *Location Aspect*, and the *Weaving* specifying the connections between the *Museum Base* and the *Location Aspect*. In Figure 8 (a), the same information is presented in form of the *aspectWebML* model editor's view.

In the following, we present details of both the *Location Aspect* and the specific *Weaving* with respect

to necessary adaptations in the *ContentModel* on the one hand (cf. Sec. 5.2.1) and in the *HypertextModel* on the other hand (cf. Sec. 5.2.2).

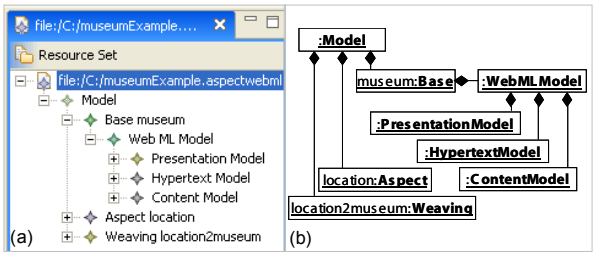


Figure 8. The location-aware *Museum* model

5.2.1 Customization in the Content Model. As in the WebML approach (cf. Sec. 5.1.1), the ContentModel needs to be extended with a *User* Entity, having an Attribute named *personalRFID* and a Relationship with the *RoomArea* Entity. This is realized using two AdpatationRules (cf. Figure 9):

a. *Content_AR1* uses SimpleAdaptation *Content_SA1* of the Aspect *Location* to introduce the *User* Entity, its *personalRFID* Attribute, and the uni-directional Relationship *user2roomArea* using the ContentModel as JoinPoint and thus, having an *enhancement* effect.

b. *Content_AR2* uses SimpleAdaptation *Content_SA2* of the Aspect *Location* to introduce uni-directional Relationship *roomArea2user* using the Entity *RoomArea* as JoinPoint and thus, having an *enhancement* effect on the Base.

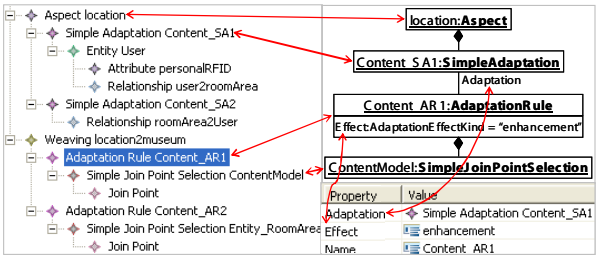


Figure 9. The location-aware content model

The reason for modeling two AdaptationRules instead of one is as follows: In WebML, every modeling concept is contained by another one, e.g., ContentModel contains Entity, which contains Relationship and Attribute. However, bi-directional Relationships are realized as a combination of two uni-directional Relationships in WebML, each being part of a *different* Entity, except for reflexive Relationships. Thus, while the *User* Entity and its contained parts, i.e., *personalRFID* and *user2roomArea*, shall be contained by the ContentModel, the *roomArea2user* Relationship shall be contained by the *RoomArea* Entity, thus resulting in two SimpleAdaptations for two different JoinPoints.

5.2.2 Customization in the Content Model. As in the WebML approach (cf. Sec. 5.1.2), we now define the necessary AdaptationRules for applying the four necessary modifications of the HypertextModel.

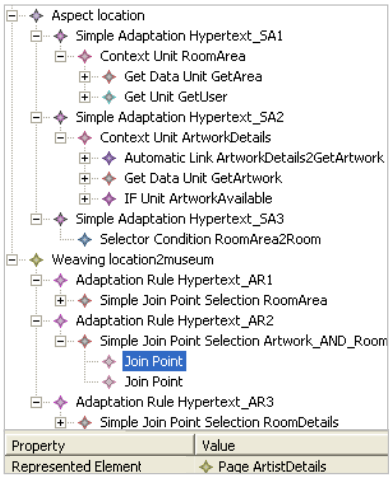


Figure 10. The location-aware hypertext model

① *Hypertext_AR1* uses SimpleAdaptation *Hypertext_SA1* to add as an *enhancement* a ContextUnit, which contains a GetUnit *GetUser* and a GetDataUnit *GetArea* to retrieve the users's location every time either the *RoomDetails* or the *ArtworkDetails* Pages are requested, to *ArtworkArea* as JoinPoint. The AdaptationRule, thus, realizes modification 1 (cf. Sec. 5.1.2).

② *Hypertext_AR2* applies SimpleAdaptation *Hypertext_SA2* to two JoinPoints, namely the *RoomDetails* and *ArtworkDetails* Pages. Thus, the rule realizes modification 2 and 4. In particular, the *enhancement* consists of a ContextUnit, which contains a GetDataUnit *GetArtwork*, to retrieve the specific *Artwork* of the *RoomArea* the visitor is currently in. Furthermore, the ContextUnit contains an IFUnit *ArtworkAvailable*, which evaluates whether a piece of *Artwork* is exhibited in the *RoomArea* and accordingly activates one of the OKLinks to either the *RoomDetails* DataUnit or the *ArtworkDetail* DataUnit.

③ *Hypertext_AR3* applies SimpleAdaptation *Hypertext_SA3* to *replace* the default SelectorCondition of DataUnit *RoomDetails*, with a SelectorCondition *[RoomArea2Room]*, thus, solving modification 3 (cf. Sec. 5.1.2).

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

5.2.3 The aspectWebML Model Editor. For the implementation of *aspectWebML*'s metamodel, we have used *Ecore*, a MOF implementation in Java, which is provided by the Eclipse Modeling Framework (EMF, http://www.eclipse.org/emf). The reason for employing Ecore was mainly the wide-spread utilization of EMF and that currently no standardized implementation of MOF 2.0 is available. Another benefit was that having an Ecore-based metamodel, we have been able to generate a tree-based model editor for *aspectWebML* using EMF's code generation facilities. The *aspectWebML* metamodel, model editor, and the *Museum* example can be downloaded from http://www.big.tuwien.ac.at/projects/aspectwebml.

6. Conclusions and Outlook

In this work, we proposed to use aspect-orientation as driving paradigm for capturing customization of ubiquitous web applications at the modeling level. We extended WebML, a domain-specific language designed for the model-driven development of data-intensive web applications, with concepts from the aspect-oriented modeling field according to our reference architecture for aspect-oriented modeling. Furthermore, we compared the original modeling approach of WebML with our *aspectWebML* approach by extending a *Museum* web application with customization functionality and reported on our prototype modeling editor.

Future work includes, first the investigation of more elaborated join point selection mechanisms, such as OCL or Join Point Designation Diagrams, and their applicability in *aspectWebML*, and second, the definition of a weaving mechanism for Aspect and Base Models in *aspectWebML*. In the long run, we intend to design a concrete syntax for *aspectWebML* and provide elaborate tool support.

7. References

[1] H. Baumeister, A. Knapp, N. Koch, and G. Zhang, "Modelling Adaptivity with Aspects", In Proc. *5th Int. Conf. on Web Engineering*, LNCS 3579, July 2005, pp. 406-416.

[2] S. Ceri, F. Daniel, M. Matera, and F. Facca, "Model-driven Development of Context-Aware Web Applications", To appear in *ACM TOIT*, 7(2), May 2007.

[3] S. Ceri, P. Fraternali, A. Bongio, M. Brambilla, S. Comai, and M. Matera, *Designing Data-Intensive Web Applicatio*ns, Morgan-Kaufmann, 2003.

[4] J. Conejero, J. Hernández, and R. Rodríguez, "UML Profile Definition for Dealing with the Notification Aspect in Distributed Environments", In Proc. *6th Int. Workshop on Aspect-Oriented Modeling*, Chicago, Illinois, March 2005.

[5] I. Garrigós, S. Casteleyn, and J. Gómez, "A Structured Approach to Personalize Websites using the OO-H Personalization Framework in Web Technologies Research and Development", In Proc. *7th Asia-Pacific Web Conference*, Shangai, China, March 2005, pp. 695-706.

[6] G. Kappel, B. Pröll, W. Retschitzegger, and W. Schwinger, "Customisation for Ubiquitous Web Applications - A Comparison of Approaches", *Int. Journal of Web Engineering and Technology*, 1(1), Inderscience Publishers, 2003, pp. 79-111.

[7] Nora Koch, "Transformations Techniques in the Model-Driven Development Process of UWE", In Proc. *2nd Model-Driven Web Engineering Workshop*, ICWE'06, Palo Alto, CA, July 2006.

[8] N. Moreno, P. Fraternalli, A. and Vallecillo, "A UML 2.0 Profile for WebML Modeling", In Proc. *2nd Model-Driven Web Engineering Workshop*, ICWE'06, Palo Alto, CA, July 2006.

[9] Object Management Group (OMG). Meta Object Facility (MOF) 2.0 Core Specification Version 2.0. http://www.omg.org/docs/ptc/04-10-15.pdf, Oct. 2004.

[10]Object Management Group (OMG), OCL Specification Version 2.0, http://www.omg.org/docs/ptc/05-06-06.pdf, June 2005.

[11]G. Ortiz, Juan Hernández, P. Clemente, and P. A. Amaya, "How to Model Aspect-Oriented Web Services", In Proc. *1st Model-driven Web Engineering Workshop*, ICWE'05, Sydney, Australia, July 2005.

[12]G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and J. Irwin, "Aspect-Oriented Programming", In Proc. *11th European Conference on Object-Oriented Programming*, 1997, pp. 220-242.

[13]A. Schauerhuber, W. Schwinger, E. Kapsammer, W. Retschitzegger, and M. Wimmer, "Towards a Common Reference Architecture for Aspect-Oriented Modeling", In Proc. *8th Int. Workshop on Aspect-Oriented Modeling*, AOSD'06, Bonn, Germany, March 2006.

[14]A. Schauerhuber, M. Wimmer, and E. Kapsammer. "Bridging existing Web Modeling Languages to Model-Driven Engineering: A Metamodel for WebML", In Proc. *2nd Model-Driven Web Engineering Workshop*, ICWE'06, Palo Alto, CA, July 2006.

[15]W. Schwinger and N. Koch, "Modeling Web Applications", In *Web Engineering - Systematic Development of Web Applications*, Wiley, 2006.

[16]D. Stein, S. Hanenberg, and R. Unland, "Expressing Different Conceptual Models of Join Point Selections in Aspect-Oriented Design", In Proc. *5th Int. Conf. on Aspect-Oriented Software Development*, Bonn, Germany, March 2006, pp. 15-26.

[17]G. Zhang, H. Baumeister, N. Koch, and A. Knapp, "Aspect-Oriented Modeling of Access Control in Web Applications", In Proc. *6th Int. Workshop on Aspect Oriented Modeling*, AOSD'05, Chicago, USA, March 2005.

Evaluating the Quality of Models Extracted from Embedded Real-Time Software

Joel Huselius, Johan Kraft,* Hans Hansson, and Sasikumar Punnekkat
Mälardalen Real-Time Research Centre, Mälardalen University, Västerås, Sweden
{joel.huselius,johan.kraft,hans.hansson,sasikumar.punnekkat}@mdh.se

Abstract

Due to the high cost of modeling, model-based techniques are yet to make their impact in the embedded systems industry, which still persist on maintaining code-oriented legacy systems. Re-engineering existing code-oriented systems to fit model-based development is a risky endeavor due to the cost and efforts required to maintain correspondence between the code and model. We aim to reduce the cost of modeling and model maintenance by automating the process, thus facilitating model-based techniques. We have previously proposed the use of automatic model extraction from recordings of existing embedded real-time systems. To estimate the quality of the extracted models of timing behavior, we need a framework for objective evaluation. In this paper, we present such a framework to empirically test and compare extracted models, and hence obtain an implicit evaluation of methods for automatic model extraction. We present a set of synthetic benchmarks to be used as test cases for emulating timing behaviors of diverse systems with varying architectural styles, and extract automatic models out of them. We discuss the difficulties in comparing response time distributions, and present an intuitive and novel approach along with associated algorithms for performing such a comparison. Using our empirical framework, and the comparison algorithms, one could objectively determine the correspondence between the model and the system being modeled.

1. Introduction

Model-based techniques such as implementation prototyping and prototype performance analysis [16] are still not widely used by industrial system developers. According to our industrial contacts, the reluctance against using model-based techniques is largely due to the initial cost of modeling of a code-oriented legacy system where models are not used today. Our research focus is on reducing this cost, thus making methods that use models more attractive in industrial settings.

Our application domain is that of real-time systems, i.e. those systems where temporal and functional correctness are equally important. The most common type of requirement for real-time systems is a bound on the *response time* for given stimuli. In a complex multitasking system, determining the bound and distribution of response times is generally difficult in practice.

In the context of this paper, a typical legacy system has all or some of the following properties: it consists of millions of lines of code, it is maintained by a large team of engineers, it contains code that originated several years ago, and it is expected to be further developed for many more years to come. Real examples of these systems can easily be found within many domains such as the automation, automotive, and telecom industries. In such systems, a large effort must be spent on keeping complexity at acceptable levels [11]. If the complexity is allowed to increase without bound, the life expectancy of these systems will be drastically reduced. Though model-based analysis can help in limiting the complexity increase, there is a reluctance of the industry to adopt these technologies. Hence, tools that ease modeling and the maintenance of models are needed. We have developed tools for *model extraction* in order to ease both modeling and model maintenance [2, 6, 7]. The models that our methods extract reflects the general behavior of the system rather than the worst-case, which is a more common focus in real-time systems research.

1.1. Contributions

In this paper, we present a framework for evaluating our proposed tools with respect to real-time properties such as the ability to accurately model response time distributions. The proposed method can also be used to compare the effectiveness of different methods of automatic model extraction for the general system behavior proposed in the literature (e.g. [8]). In addition to the framework, a intuitive and novel method for comparing time distributions is introduced. We supply the method as well as a set of algorithms to perform

*Formerly Johan Andersson.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

the comparison. In the framework, the comparison method plays a critical role as it provides a measurement that can be used to evaluate the performance of a model with respect to the system.

1.2. Organization

The remainder of this paper is organized as follows: Section 2 provides background on previous work and our problem domain. Section 3 presents framework for empirical testing and comparison of proposed methods for automatic model extraction. Section 4 provides a definition of an objective measurement to compare distributions of response times, and a set of algorithms to perform measurements. Section 5 concludes the paper.

2. Background

In our work, we are developing tools for automatic or semi-automatic modeling of legacy real-time software. The models are intended to be used in model-based implementation prototyping and prototype performance analysis. As a part of this effort, we have developed a unified method for dynamic model extraction [6, 7]. The basic idea is to input execution recordings of the legacy system that is to be modeled, covering context switches, inter-process communication (IPC), and updates of important variables. If the recordings contain enough information, a tool implementing the unified method automatically delivers a validated model, otherwise the user is advised to alter monitoring or extend recordings. The unified method consists of two separate methods; one for *automatic model generation* and one for *automatic model validation*. Model extraction is performed separately for each task, and a collection of models for all tasks in the system can be merged and used to analyze the system by means of simulation. A set of models produced by the method can be used to prototype future design options with respect to response time requirements and functional behavior. A case study has been performed on a state-of-practice industrial robot system to show the applicability of the method [7].

In a parallel effort, we are also developing tools for semi-automatic static model extraction based on implementation code and execution recordings [2]. One of our future aims is to compare the performance of these two different strategies.

2.1 Automatic model generation

Introduced in [6], our automatic model generation can, based on a set of recordings of a running system, output a model of the system. The recordings cover system level events such as context switches and communication. Optionally data state manipulation on task level is included, allowing modeling of causal relations. Generation is performed in three stages: First, an event sequence for each task is extracted from each recording. Second, all event sequences for each task are merged into a tree structure. Third, each tree is translated into a model for the task.

To express abstraction from the implementation, the models contain probabilistic elements: Selections can be made based on probabilities (or on data state), and execution time requirements can be described as probability distributions. For the effective use of the modeling language, there is a tool-suite for simulating and analyzing the performance of models.

A limitation of this method is that it does not model loops within tasks as in [3]. However, our assessment is that such loops are often avoided in embedded real-time systems due to predictability requirements. Also, this limitation does not effect the contributions of the paper: the evaluation framework and the comparison of sampled distributions.

2.2 Automatic model validation

We validate that the recordings used to generate the model are sufficient to describe the system by answering the question "Would the model be drastically better if the length or number of recordings used during model generation were increased?". Automatic model validation, introduced in [7], uses a set of system execution recordings to answer that question. The model and the recordings are transformed into a set of communicating timed automata with integer variables [1, 4]. While the model-automaton is a graph structure which may contain more than one transition from each label, the recording-automata are all sequential with one or zero transitions from each label. The validation is performed by reachability analysis of the final state in each recording-automaton when co-simulated with the model-automaton.

To allow the model to be an approximate abstraction of the system, the recording-automata are constructed using a *leeway*-parameter. The higher the leeway, the more forgiving the recording-automaton will be. The maximum allowed leeway can be supplied by the user as a parameter.

The stopping criteria of the validation is based on two factors: The *completeness measure*, i.e. the probability that the model can replicate any job that the system can exhibit, and the *accuracy measure*, i.e. the relation between the probability that the system exhibits a particular job and the probability that the model exhibits an equivalent job.

Validation can provide the maximum required leeway, the completeness measure, and the accuracy measure as auxiliary output. The primary output is the binary answer to the question posted at the top of this section.

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

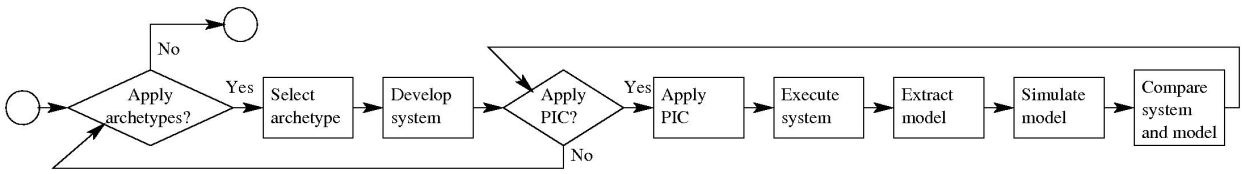


Figure 1. A process-view of the framework for empirical evaluation.

Together, model generation and model validation forms model extraction.

2.3 Response time

In real-time systems, the temporal and behavioral requirements are equally important. We assume that the embedded real-time systems we consider consist of a set of *tasks* that can either be event or time triggered. As a task is triggered, a *job* (a task invocation) is executed for some period of time, after which the task will await until further triggering. Two jobs of the same task cannot overlap. We label the time measured from the point in time where the job is triggered (the *release time*) until the end of a job the *response time* of the job.

3. Framework for empirical evaluation

In this section, we introduce the framework for empirical evaluation depicted in Figure 1. The purpose of the framework is to evaluate the effectiveness and general applicability of dynamic model extraction. We use a notion of archetypes to describe architecturally different system designs. For example, a system consisting of a set of periodic tasks without inter-process communication (IPC) is a different archetype than a system consisting of event triggered tasks where the exchange of messages trigger execution.

We have implemented an instance of the framework, where the system platform is a multitasking, fixed priority scheduled, instruction-set simulator that we have developed specifically for this purpose. A system of tasks is defined by a system definition file together with an assembler-file per task. Each task is either defined as triggered periodically or triggered as a result of input on an IPC-queue. The simulator can handle 16 types of instructions, including absolute and relative sleep, branching, IPC, explicit logging, register manipulation and testing, and random number generation. Each instruction takes one clock cycle to complete. As the system is executed, occurred task switches, performed IPC, and executed explicit log instructions are automatically recorded in a system specific log.

To each system definition, we can add an increasing portion of *Population*, *Imperfectness*, or *Complexity* (PIC) for each test performed on the system:

- *Imperfectness* regards the quality of the data-state recording in the system. With low imperfection, all relevant information is recorded. With high imperfection, some data state information is omitted, which leads to that the execution time distributions for a given data state are non-trivial.
- *Complexity* regards the task complexity, e.g. the number of tasks in the system, the nature of environment stimuli, and the number of data states.
- Finally, *Population* regards system wide issues such as the number of tasks in the system as well as the recording lengths and recording set sizes.

For each system definition and PIC combination, we perform two sets of simulations of the system: recordings from the first set is used to generate the model and the second set is used to validate the model (as described in Section 2). The intention is that as PIC changes the system, the model should follow and be affected correspondingly. Varying the PIC will test the robustness of the model extraction with respect to changes in the system – this form of robustness is imperative for successful implementation prototyping [13].

In our implementation of the framework, we compare the system and model by analyzing response time distributions from both system and model. After model extraction, the model is simulated and compared to new simulations from the system. The collected set of comparisons is the output of the framework implementation. These can then be analyzed to verify that the method of model extraction performs sufficiently well, or even to compare several methods of model extraction. As the set of archetype-PIC combinations is intended to be large, the comparison should be automated. We present a novel automated measurement in Section 4 that can be used in such a comparison.

3.1. Archetypes and PIC

The following are examples of archetypes that are used in our study:

1. **Client-server without reply.** This archetype describes a common design pattern in the industrial systems that

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

we have encountered. A client sends varying service requests to a server that services the requests. Results of the computations may effect the environment or successive requests to a third or fourth task. PIC applied are priority ordering, frequency increase, and execution time increase.

Specifically, this archetype is implemented with two tasks T_1 and T_2. With a fixed periodicity, Task T_1 sends a message to Task T_2 that reacts on the contents of the message. We distinguish between four different contents, representing four different commands, plus one default behavior in the case that the content in unrecognized.

2. **State machine.** Here, a task acts as a state machine which makes one transition per job. Transitions are triggered by messages from the environment or from another task. Task mode changes can be expressed by this archetype. In contrast to the client-server archetype, the same message can trigger different behavior at different points in time. PIC applied are reduced recording of variable assignments (simulating poor probing), environment stimuli, complexity of the state machine, priority ordering, frequency increase, and execution time increase.

   The archetype is implemented by two tasks T_1 and T_2. With a fixed periodicity, Task T_1 sends a message to Task T_2 with randomly selected contents 0 or 1. The event triggered Task T_2 consists of a finite state machine that can make one state transition per job. The target state of each transition is depending on the contents of the triggering message from T_1. A variable is maintained to keep track of the current state.

3. **Purely periodic without communication.** A task set of periodic tasks where execution times for any given task varies randomly between jobs within determined intervals. In this case, the PIC consists of increase of the task set size.

   In our experiments, the implementation consists of at most seven periodic tasks T_{1-7}, that execute a bounded random interval in each job. For each task, the worst case execution time (WCET), the period (T), utilization (U), and analytical worst case response time (R) are described in Table 1.

4. **Feedback loop.** Here, tasks exchange messages in a loop. Examples include client-server with reply, or a feedback control system. The PIC consists of priority ordering, message complexity, message reply frequency, and environment stimuli.

   The implementation consists of five tasks, two of which (T_1 and T_2) are implementing the feedback loop, and the three remaining are concurrently executing a client-server without reply and a simple periodic task.

	WCET	T	R	U
T_1	10	80	10	12.5%
T_2	30	120	40	25.0%
T_3	20	160	60	12.5%
T_4	15	180	75	8.3%
T_5	30	200	115	15.0%
T_6	40	300	300	13.3%
T_7	80	1000	960	8.0%
Σ				94.6%

Table 1. Maximum utilization for Archetype 3.

5. **State machine feedback loop** This archetype is a combination of archetypes 2 and 4, as is the PIC.

   The implementation consists of two tasks T_1, and T_2. Both tasks are state machines, Task T_1 generates input to trigger Task T_2, Task T_2 generates input that, if available, will affect the execution of Task T_1.

4. Comparison of sampled time distributions

The framework proposed above assumes that it is possible to objectively compare a system with a model of that system. We have chosen to implement this by comparing distributions of e.g. response times from the system and the model. However, methods known to us from literature prove unintuitive in this setting:

The Euclidean distance metric and the χ^2 test of independence [5] are both *categorical* in the temporal dimension, which results in that they are not sensible to the difference that two samples have almost the same response time if they are in different categories. They are only sensible in the sample dimension, which means that they can acknowledge that *almost* the same number of samples in both distributions have response times in the same category. This leads to unintuitive results due to false negatives.

The Kolmogorov-Smirnov test [12] assumes that one of the distributions in a comparison is mathematically modeled [10, 14]. However, the execution time distribution of a program is often very complex. On the source code level in a system implementation, selections where one path has a significantly longer execution time than the other are common. Execution time distributions that cover both legs of such selections does not follow a simple pattern. Therefore, it cannot be assumed that a response time can be classified to a known distribution (e.g. a normal distribution). We know of no universal method of determining or estimating similarity between unclassified finite discrete distributions.

To amend this lack of a suitable method of comparison, we introduce a novel objective measurement for sampled distributions based on the two notions of *divergence* (see Definition 1) and *difference* (see Definition 2).

Definition 1 (divergence)**.** Let U be the set of samples. There is a function $time : U \rightarrow \mathbb{Z}^*$. For a given sample $u \in U$, we use $time(u)$ to denote the value of that sample.

Let $A, B \in 2^U$ be two sets of samples from two sources (e.g. a model and a system) with equal cardinality. We define a *match* between these two as a *bijective* mapping between A and B, $\delta_{AB} : A \rightarrow B$. Let C_{AB} be the full set of matches (i.e. the full set of bijective mappings) between A and B.

We are now able to define a measurement of a match $\delta_{AB} \in C_{AB}$ by the function $[\,] : C_{AB} \rightarrow \mathbb{Z}^*$ as:

$$[\delta_{AB}] = \max_{a \in A} |time(\delta_{AB}(a)) - time(a)|$$

Then, the *divergence* between two sets of samples is the measurement of the most favorable match in the sense that the measurement is minimized:

$$\text{divergence}(A, B) = \min_{\delta_{AB} \in C_{AB}} \{[\delta_{AB}]\}$$

□

Intuitively, for two equally sized sampled distributions A and B, δ_{AB} describes a mapping from each sample in A to a unique sample in B (the uniqueness follows from that the mapping is bijective). *Divergence* is then considering the best possible mapping in the sense that the largest difference of response times that the matched samples represent should be as small as possible. In the following sections, we will present algorithms for measuring the divergence between two distributions.

Definition 2 (difference)**.** Given Definition 1, the *difference* between two sets of samples with equal cardinality is defined by:

$$\text{difference}(A, B) = |\{a \in A \mid time(\delta_{AB}(a)) \neq time(a)\}|$$

□

Intuitively, for a mapping of samples, *difference* counts the number of mapped samples whose response times are not equal.

Then, with divergence C_{min} and difference D, the comparison between distributions A and B is defined by the tuple $cmp(A, B) = \langle \frac{C_{min}}{C_{max}}, \frac{D}{lcm(|A|,|B|)} \rangle$, where C_{max} is the difference between the smallest and the largest samples from both distributions.

The relation between divergence and difference is used to quantify the correspondence between two sampled distributions as described in Figure 2. If the relations between divergence and difference is in an area with same shade of gray, two distributions are considered similar. The two are more similar the darker the shade is. As explained by the figure, if both divergence and difference are small, the distributions are very similar, if the divergence is small but the difference is large, the distributions are similar, etc. In the figure, we have exemplified four comparisons between imagined distributions. Distribution A is compared to distributions B_1, B_2, B_3, and B_4 respectively. The relation between comparisons *cmp*(A, B_1) and *cmp*(A, B_4) tells us that B_1 is more similar to A than B_4 is, since the former lies in a darker area than the latter. Analogous, *cmp*(A, B_2) and *cmp*(A, B_3) tells us that B_2 and B_3 are equally similar to distribution A, since they lie in an area with same shade.

The plot of Figure 2 is conceptual, the relations between divergence and difference must be defined for a given use of the comparison. If two methods of model extraction are compared in the framework above, the divergence-difference relation is defined once and used throughout the comparison. The intention is that the divergence-difference relation should reflect the quality that is required of the models. This is defined by the intended use of the models and the application domain of the system.

In the case of performing this measurement on response times, the observability problem [15] must be respected: Take the example of a system with a set of strictly periodic tasks. Here, especially if the system load is high, it is likely that jobs of tasks are ready to execute long before they receive their first quanta of processing time. According to our definition of response time (see Section 2.3), the time of the

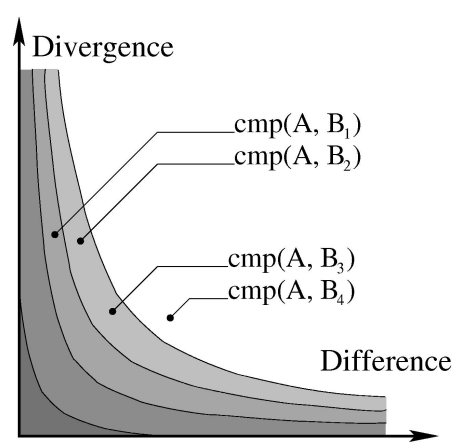


Figure 2. Conceptual divergence and difference plot.

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

triggering of the task must be known. Thus, probes that can access the ready queue of the operating system must be used to obtain a truthful measure of the response time.

4.1. Divergence and difference algorithms

We have implemented algorithms for calculating the divergence and difference as defined above. The following terminology will be used:

A distribution, or series of measured response times, d is represented as a binary tree where each node has the attributes *value* and *count*. The values of the attributes of node n of d are referred to with a doted notation (e.g. *n.value*). So are the siblings of n (e.g. *n.left*). The distribution has two operations *add* and *remove*, these are called using a similar doted notation (e.g. *d.add(e)*, where e is a response time). An *element* is a response time measurement represented in a series of response times.

A stack s has operations *push*, *pop*, *top* and the attribute *size*, which are called with a doted notation (e.g. *s.pop()*). The attribute is transparently manipulated by the operations *push* and *pop* to reflect the number of elements on the stack. The function *top* is used to investigate the topmost element on the stack without removing the element. The stack will be used to store elements on the form $\langle e_1 \in \mathbb{Z}^*, e_2 \in \mathbb{Z}^*, alternative \in \{true, false\}\rangle$, where the first element e_1 of the tuple is a response time from the first distribution, the second element e_2 is a response time from the second distribution, and the third element *alternative* denotes whether there are other feasible matches for the pair formed by the first two elements. To refer to the elements in the topmost entry of the stack, we use a doted notation (e.g. *s.top().alternative*).

To perform the comparison, the sizes of the two series of response times are first normalized by linearly adjusting the *count* of *value*'s such that the total sum of *count*'s in each of the two trees is equal to the Least Common Multiple (LCM) of the original total sum of *count*'s of both trees.

Then, the divergence C_{min} and the difference D for the comparison $cmp(d_1, d_2)$ are defined as follows: C_{min} is the smallest C such that $IsDivergence(d_1, d_2, C)$, as defined in Algorithm 1, evaluates to true. D is the size of the difference between the two distributions as defined by Algorithm 4.

4.2. Algorithms for measuring the divergence of two distributions

Algorithm 1 matches elements in two distributions so that the values of each match are less than or equal to a specified relative divergence. If all elements in the two distributions are successfully matched, the divergence between the two distributions is less than or equal to the specified relative divergence. In this process, Algorithm 1 uses Algorithm 2 to find the lowest value with a count larger than zero in a specified distribution. In turn, Algorithm 2 uses Algorithm 3 to decide which of two values is the smallest value in a given distribution.

Intuitively, if both values are in the specified distribution, Algorithm 3 returns the smallest of these. Otherwise, if only one value is in the specified distribution, that value is returned and the return value is deemed *valid*. If none of the values are in the distribution, *invalid* is returned.

Algorithm 2 compares the value of the current node, if the count of that value is larger than zero, to the valid results of recursive calls for the left and the right siblings of the binary tree. If the count of the value is less than or equal to zero, only the valid results of the recursive calls are considered. Out of all valid values, Algorithm 3 is used to find the smallest value within the specified distribution. This smallest value is returned and the return value is deemed *valid*.

Algorithm 1 $IsDivergence(d_1, d_2, C)$ compares two response time distributions d_1 and d_2 represented as binary trees with respect to divergency $C \in (0, C_{max})$.

Require: $srb.size = 0 \wedge d_1.size = d_2.size$
$cMin := C$
repeat
  $e_1 := FindLow(d_1, 0, \infty, 0)$
  $e_2 := FindLow(d_2, e_1, C, cMin)$
  if e_2 is valid **then**
    $d_1.remove(e_1)$
    $d_2.remove(e_2)$
    $e_2^{alt} := FindLow(d_2, e_2 + 1, C + (e_1 - e_2), 0)$
    if e_2^{alt} is valid $\wedge$ $(e_1 \neq srb.top().e_1 \vee e_2 \neq srb.top().e_2)$ **then**
      $srb.push(\langle e_1, e_2, true\rangle)$
    else
      $srb.push(\langle e_1, e_2, false\rangle)$
    end if
    $cMin := C$
  else if $(srb.size > 0)$ **then**
    repeat
      $d_1.add(srb.top.e_1)$
      $d_2.add(srb.top.e_2)$
      $popped := srb.pop()$
    until $(srb.size \leq 0 \vee popped.alternative = true)$
    if $popped.alternative = true$ **then**
      $cMin := popped.e_1 - popped.e_2 - 1$
    end if
  else
    return false
  end if
until $d_1.size \leq 0$
return true

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

Algorithm 2 $FindLow(d, value, cMax, cMin)$ finds the lowest value in the binary tree d, with attributes *value* and *count*, such that $d.value$ is in the interval $(value - cMin, value + cMax)$ and $d.count$ is larger than zero.

```
Require: cMax ≥ cMin
min := invalid
if d is valid then
    if d.count > 0 then
        min :=
            MinInInt(d.value, min, value, cMax, cMin)
    end if
    a := FindLow(d.right, value, cMax, cMin)
    if a is valid then
        min := MinInInt(a, min, value, cMax, cMin)
    end if
    b := FindLow(d.left, value, cMax, cMin)
    if b is valid then
        min := MinInInt(b, min, value, cMax, cMin)
    end if
end if
return min
```

Algorithm 3 $MinInInt(a, b, value, cMax, cMin)$ returns the lowest of a and b which is in the interval $(value - cMin, value + cMax)$, if any. Otherwise, the return value is invalid.

```
if a ∈ (value − cMin, value + cMax) then
    if b ∈ (value − cMin, value + cMax) then
        if a < b then
            return a
        else
            return b
        end if
    else
        return a
    end if
else
    if b ∈ (value − cMin, value + cMax) then
        return b
    else
        return invalid
    end if
end if
```

If the algorithm is unable to find a value in the distribution, *invalid* is returned.

For a given divergence, for each count of each value in the first distribution, Algorithm 1 attempts to find a match in the second distribution. A match results in that the count of the matched value in both distributions is decreased, and the match is pushed on a stack together with information (*alternative*) of whether if there are other potential candidates for the particular match. If the search for a match fails, the stack is popped and the distributions are consequently repopulated until an old match with potential for another match is popped. Then the process is resumed. If, during that rollback process, the stack becomes empty, the match is said to have failed.

Pushed on the stack are a tuple, where one element (*alternative*) describes if the popping of the element on the stack will lead to that new untried combinations of matches can be performed. The value of *alternative* is decided by two criteria tested on Line 9 of Algorithm 1: First, the result of the call to Algorithm 2 on Line 8 of Algorithm 1, where the second of the two distributions are searched to determine if other combinations exists for the chosen element of the first distribution. Then, by checking the top of the stack to determine if a similar match as the one about to be made has been made just recently. The second criteria is important in terms of computational complexity; with that criteria, many unnecessary tests are avoided, but it is strictly speaking not important for the operation of the algorithm.

In the test, if the divergence C is large enough, all elements in the first distribution can be matched with any element in the second distribution. From understanding Algorithm 1, we see that such a situation must lead to that $IsDivergence$ returns *true*. The smallest value of this largest C, denoted C_{max}, when $IsDivergence$ must return *true* is equal to the difference between the largest observed response time and the smallest observed response time.

Plotting the result of $IsDivergence$ with two given distributions as a function of C, we get a monotonous function which starts at *false* and subsequently, at some value of C, turns *true*. As the function is monotonous for two given distributions we can use binary search [9] in the interval $C \in (0, C_{max})$ to find the smallest divergence C_{min} that satisfies Algorithm 1. Seen in relation to the size of the interval $(0, C_{max})$, C_{min} provides an objective measure on the likeness of the two distributions.

The use of a stack in the realization of this algorithm is essential to the implementation due to the potentially large memory requirements. As the distributions in the measurement are normalized by size, the total sum of the counts of values is potentially large, which will inflict on the number of matches that need to be identified. Storing this large number of matches will require large amounts of memory, but the use of a stack provides us with the opportunity to reduce the amount of physical memory used without large penalty to the execution time: One of the properties of a stack is that, at any given time, only the topmost element is needed. Thus, as the order of element usage is, if not determined, then at least restricted, a large part of the stack can be written to secondary storage (i.e. to file). We have implemented this by defining a threshold for the maximum

number of elements that are allowed in primary memory, when this number is reached, the stack is flushed to file.

The theoretical worst case complexity of the *IsDivergence* algorithm is $O(N!)$, where N is the size of the distribution sizes when normalized by size. As N is likely to be large, the worst-case complexity is very high. However, due to the *alternative* field in elements on the stack, it is unlikely that the worst-case ever occurs. Typically, with N of approximately $1,000,000$, the algorithm takes in the order of minutes to execute.

4.3. Algorithms for measuring the difference of two distributions

The difference of two series of response times is computed as follows: Algorithm 5 finds, if any, the node in the binary tree representation of a distribution that has a given value. Algorithm 4 use Algorithm 5 to calculate the size of the difference between two distributions, which are normalized by size. Intuitively, the algorithm counts all elements of the first distribution that have no corresponding element in the second distribution.

Algorithm 4 $Difference(d_1, d_2)$ calculates the difference between the distributions d_1 and d_2.

```
s := 0
if d1 is valid then
    s := Difference(d1.right, d2)
    s := s + Difference(d1.left, d2)
    d := Find(d2, d1.value)
    if d is valid then
        if d1.count − d.count > 0 then
            s := s + d1.count − d.count
        end if
    else
        s := d1.count
    end if
end if
return s
```

Algorithm 5 $Find(d, value)$ implements binary search [9] to find the node in the distribution d that matches the value *value*.

```
if d is valid then
    if d.value = value then
        return d
    else
        if d.value > value then
            return Find(d.left, value)
        else
            return Find(d.right, value)
        end if
    end if
end if
return invalid
```

The theoretical complexity of this algorithm is $O(PQ)$, where P is the number of unique sample values in the first distribution, and Q is the number of unique sample values in the second distribution.

4.4. Example

To exemplify, for a given task, assume that a series $\langle 1,1,1,1,1,2,5,6\rangle$ of response times has been observed in the system, and that the series $\langle 1,2,3,5\rangle$ has been observed in the model. The LCM of the number of elements in the series (8 and 4) is 8. Thus, normalization of the size of the series gives that the series observed at the system remain unchanged, and the series of response times observed at the model are $\langle 1,1,2,2,3,3,5,5\rangle$. In this example, we will not use binary search, but to illustrate the algorithm pick values of divergence that gives interesting executions of the algorithm.

After normalization, the series of system response times $\langle 1,1,1,1,1,2,5,6\rangle$ are represented as the binary tree d_1, and the series of model response times $\langle 1,1,2,2,3,3,5,5\rangle$ are represented as the binary tree d_2.

In our search for the measure of divergence, performed in the interval $(0,5)$, we start with divergence value $C = 0$. In Algorithm 1, Line 3 will set $e_1 = 1$. At Line 4, e_1 is assigned 1. At Line 9, the first criteria is evaluated to false because there is no other match when the divergence is zero. The second criteria will also evaluate to false because the stack is empty. Thus, the tuple $\langle 1,1,false\rangle$ is pushed on the stack.

When the match progressed such that two matches have been found, distribution d_1 has two occurrences of the value 1 remaining, while distribution d_2 has none, the search will rollback to find a new path. However, none of the matches found has had any alternative matches, hence the test with divergence $C = 1$ will fail.

If $C = 1$, the first two matches will match value 1 from d_1 with 1 from d_2. Then, a third and fourth match will find value 1 from d_1 and 2 from d_2, leaving the distributions as follows: $d_1 \equiv \langle 1,2,5,6\rangle$ and $d_1 \equiv \langle 3,3,5,5\rangle$. The next search for a match to $e_1 = 1$ will fail, and a rollback will commence.

It is then discovered that the third match, performed between 1 and 1, could also have been performed between 1 from d_1 and 2 from d_2. This satisfies the first criteria of Line 9. The second criteria is also satisfied as the previous match performed was between 1 and 1. Thus, the rollback stops, and the next found match is the aforementioned alternative match between values 1 from d_1 and 2 from d_2.

Subsequently however, also this divergence will prove to

conclude a failure. It is not until the divergence is higher than or equal to three ($C_{min} = 3$) that the algorithm finds matches for all samples in the distribution. In relation to the span of the values in the distributions, which is 5, this is a relatively large divergence.

The difference of the two distributions is computed using Algorithm 4 in the form $Difference(d_1, d_2)$. The algorithm concludes that the difference is four ($D = 4$). In relation to the size of the distributions when normalized by size, which is 8, this is a relatively large difference.

The result of the comparison $cmp(d_1, d_2)$ is $\langle \frac{3}{5}, \frac{4}{8} \rangle$. Interpreting these measurements in the light of Figure 2, we conclude that the distance between the distributions is not negligible. For example, if the second series of observations would have been $\langle 1, 1, 2, 5 \rangle$, the conclusion would have been different.

5. Conclusion

We have presented a framework for evaluating the feasibility of using the automatic model extraction approach to obtain models of real-time software. We have presented examples of generic and commonly used archetypes and described their role in the framework with respect to PIC. The method has the same scalability issues as has any other application of testing: it may take time to construct and implement archetypes and PIC as well as to perform an evaluation. However, archetypes and PIC are generic and can be reused by other evaluations, and a method needs only to be evaluated once (evaluations of different methods can then be inter-related). Once a library of archetypes and PIC has been constructed, evaluation is essentially an automated task. Also, as the evaluation needs only to be performed once per method, our opinion is that the complexity of performing the evaluation is acceptable.

Further, we also discuss the fundamental issues of how to compare models against systems and we provide an objective measurement that solves this problem within the context of comparing response times of models and systems.

In our future work, we plan to explore and identify more relevant archetypes as well as perform evaluation of our method for dynamic model extraction. We are also developing more efficient algorithms for measuring the divergence, and are investigating other potential comparison methods.

5.1. Acknowledgments

We would like to acknowledge the fruitful discussions we had with Stefan Bygde in formulating the definition of divergence and Dr. Gustav Öquist in formulating the evaluation framework. We also thank the anonymous reviewers for their valuable comments.

References

[1] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, April 1994.

[2] J. Andersson, J. Huselius, C. Norström, and A. Wall. Extracting simulation models from complex industrial real-time systems. In *Proceedings of the International Conference on Software Engineering Advances (ICSEA)*, October 2006.

[3] D. Angluin. Learning regular sets from queries and counterexamples. *Information and Computation*, 75(2):87–106, November 1987.

[4] J. Bengtsson, W. O. D. Griffioen, K. J. Kristoffersen, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Automated analysis of an audio control protocol using uppaal. *Journal of Logic and Algebraic Programming*, 52-53:163–181, July-August 2002.

[5] D. Freedman, R. Pisani, and R. Purves. *Statistics*. W. W. Norton & Company, 3rd edition, 1998.

[6] J. Huselius and J. Andersson. Model synthesis for real-time systems. In *Proceedings of the 9th European Conference on Software Maintenance and Reengineering*, pages 52–60, March 2005.

[7] J. Huselius, J. Andersson, H. Hansson, and S. Punnekkat. Automatic generation and validation of models of legacy software. In *Proceedings of the 12th International Conference on Embedded and Real-Time Computing Systems and Applications*, pages 342–349, August 2006.

[8] P. K. Jensen. *Reliable Real-Time Applications. And How to Use Tests to Model and Understand*. PhD thesis, Aalborg University, Denmark, February 2001.

[9] D. E. Knuth. *The Art of Computer Programming*, volume 3. Sorting and searching. Addison-Wesley, 1973.

[10] D. E. Knuth. *The Art of Computer Programming*, volume 2. Seminumerical algorithms. Addison-Wesley, 2nd edition, 1981.

[11] M. M. Lehman. Laws of software evolution revisited. In *Proceedings of the 5th European Workshop on Software Process Technology*, pages 108–124, October 1996.

[12] F. J. Massey, Jr. The Kolmogorov-Smirnov test for goodness of fit. *Journal of the American Statistical Association*, 46(253):68–78, March 1951.

[13] A. K. Mok. Tracking real-time systems requirements. In *Proceedings of the 7th International Conference on Real-Time Computing Systems and Applications*, pages 333–336, December 2000. Invited Address.

[14] NIST/SEMATECH. e-handbook of statistical methods. Internet URL http://www.itl.nist.gov/div898/handbook/, November 2006.

[15] W. Schütz. Fundamental issues in testing distributed real-time systems. *Real-Time Systems*, 7(2):129–157, September 1994.

[16] A. Wall. *Architectural Modeling and Analysis of Complex Real-Time Systems*. Phd thesis no. 5, Mälardalen University, Sweden, September 2003. Available at: www.mrtc.mdh.se.

Visualisation of Domain-Specific Modelling Languages Using UML

Bas Graaf
Delft University of Technology
The Netherlands
b.s.graaf@tudelft.nl

Arie van Deursen
Delft University of Technology and CWI
The Netherlands
arie.vandeursen@tudelft.nl

Abstract

Currently, general-purpose modelling tools are often only used to draw diagrams for the documentation. The introduction of model-driven software development approaches involves the definition of domain-specific modelling languages that allow code generation. Although graphical representations of the involved models are important for documentation, the development of required visualisations and editors is cumbersome. In this paper we propose to extend the typical model-driven approach with the automatic generation of diagrams for documentation. We illustrate the approach using the Model Driven Architecture in the domains of software architecture and control systems.

1. Introduction

Model-driven engineering (MDE) refers to software development approaches in which models are considered the primary development artefacts [1] (instead of source code). In these approaches software models are gradually transformed (automatically) into source code by means of model transformations. Additionally, such models are used for other (automated) software engineering tasks, such as performance analysis.

Typically, MDE approaches are based on modelling languages that offer abstractions focussed on a particular domain. Such languages are referred to as domain-specific modelling languages (DSMLs). From DSML-models code is generated for a particular software platform. DSMLs have been developed for various types of domains, such as software engineering (e.g., software architecture [2]) and application domains (e.g., insurance products [3]).

In general, the use of DSMLs has clear advantages over the use of general-purpose languages [4]. More in particular, in the context of MDE, our experience in industrial case studies [5, 6] indicates that the use of a general-purpose language, such as UML, leads to (unnecessary) complex model transformations, for instance to generate code. As such, the introduction of MDE typically requires the development of DSMLs.

Although mechanisms are available to define and implement the abstract syntax of DSMLs, such as the MetaObject Facility (MOF [7]) and the Eclipse Modeling Framework (EMF [8]), not much support is available for the definition their graphical notation (concrete syntax). As a result development of adequate graphical editors and visualisations requires considerable effort.

For some software engineering tasks, such editors are not required. For instance, developers can use a textual syntax for the creation of models that can subsequently be processed by model transformation tools. However, other tasks, such as documentation, do require some form of graphical representation. It is this problem that motivates this paper.

The basic idea of this paper is simple: when devising a new DSML we try to leverage existing visual notations and modeling tools. We propose to expand the typical MDE process in which abstract models are gradually transformed into code, with (partial) generation of documentation. To this end we combine the use of DSMLs for code generation and other automated software engineerings tasks, with the use of UML for documentation. The approach uses model transformations to specify the mapping between DSMLs and UML. The diagrams corresponding to those models, as visualised by off-the-shelf UML tools, are used in the documentation. To investigate the arguments for and against this idea, we study how

- this approach works for various architectural views;
- UML can be used as the target language for visualising these views; and
- model transformations can be used to specify and automate the mapping.

In practice, the extra effort required for the development of graphical editors can hamper the introduction of MDE. Consider the following scenario. A software development organisation decides to introduce MDE. Currently, the developers use UML. However, as in many other organisations, they only use UML modelling tools for *drawing* diagrams [9]. These diagrams are important for the communication with other stakeholders, as they

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

constitute an essential part of the documentation. The introduction of MDE involves the definition of DSMLs from which code will be generated. Furthermore, as the developers are comfortable with using a textual syntax for these DSMLs, no graphical editors are developed. The result is that they now have to create DSML models for code generation as well as UML diagrams for documentation. Considering the current use of UML, as investigated by Lange et al. [9], and the upcoming of MDE approaches, such as the Model Driven Architecture (MDA, [10]), this is not an unlikely scenario.

We investigate the feasibility of our approach in the domain of software architecture. In Section 2 we introduce the languages specific to this domain, and the standard documentation approach. Our approach for the model-driven documentation of software architecture, MDAV, is presented in Section 3 and we report on a small case study in Section 4. The approach is easily applied to other domains as well. An additional (industrial) case study involving a different type of models is presented in Section 5. We discuss the benefits and limitations of the approach in Section 6. After discussing some related work in Section 7, we conclude with an overview of our contributions and opportunities for future work in Section 8.

2. Background

In this section we introduce modelling and documentation in the domain of software architecture. Furthermore, we discuss some of the technologies that enable our approach.

2.1. Software architecture

Modelling Several notations have been developed to specify architectural models. These architecture description languages (ADLs, see [2] for an overview) mostly consider an architecture to be a configuration of runtime components and connectors.

Due to their formal syntax and semantics ADLs enable automatic code generation and analysis. Despite these benefits, and although ADLs have received much attention from the architecture research community, they have not been applied much in industry [11].

Although UML is aimed at object-oriented modelling, it allows practitioners to address a wide range of issues [12]. Therefore, and because of the availability of supporting (graphical) modelling tools, it is often used in practice to describe software architectures [13, 9]).

A drawback UML is the semantic mismatch between architectural and UML's OO concepts, resulting in compromises between completeness and legibility [14]. Furthermore, for automatic processing of models (e.g., for code generation) the complexity of UML results in complex model transformations [5, 6].

Documentation Because in industrial practice a software architecture is too complex to describe in a single stroke different views are used for its documentation. Different types of views have been defined to address specific concerns. The two most prominent categories of architectural views are module views and component-and-connector (C&C) views [15].

A module view addresses the question of how a system is *developed*; it defines the most important implementation units (modules) and their relations. Module views are used, for instance, to evaluate the maintainability of a system as implied by its architecture.

A component-and-connector (C&C) view, on the other hand, addresses the question of how a system *works*. It describes a system in terms of runtime components and connectors. A component is an abstraction of a computational element; a connectors is an abstraction for the way components interact. As such, a C&C view is more suited for analysis of runtime properties, such as performance.

More specific types of views are defined by imposing restrictions on the type of elements and relations allowed in a view. In a module-uses view, for instance, only 'uses' relations are allowed.

In the terminology of IEEE Std 1471-2000 [16], a view conforms to a viewpoint that "specifies the conventions for using and constructing a view". A viewpoint addresses a set of stakeholder concerns. A number of viewpoint sets is available from literature, such as [15]. Furthermore, in practice also custom viewpoints are defined. Typically, a viewpoint definition prescribes a modelling language or notation that enables the specification of an architectural model that addresses the concerns of the viewpoint. As an example, a C&C viewpoint might refer to a particular ADL. In summary, a viewpoint defines a type of views and a view is a particular representation of a particular system.

In practice the architectural model for a view is primarily used as a figure or diagram (the view's primary presentation [15]) in a document that describes the view. Because of their wide-acceptance and available tool support often UML diagrams are used for this [13].

2.2. Enabling MDE technologies

Our approach for model-driven documentation is based on model transformations. This requires capabilities for (meta)modelling, model transformation, and model interchange.

For the definition of metamodels we use the MetaObject Facility (MOF [7]). An implementation of

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

the MOF is available as an Eclipse plugin, the Eclipse Modeling Framework (EMF [8]).

The EMF plugin generates an implementation for a metamodel as a set of Java classes that offers an interface that allows developers to manipulate conforming models. These models can be serialised to an XML document using XML Model Interchange (XMI [17]). Additionally, a simple tree-based editor is generated that can be used as an Eclipse plugin for the creation and inspection of associated models. As an example consider the screenshot of such an editor in Figure 3(b). This editor is also capable of validating a model against its metamodel.

The Atlas Transformation Language (ATL [18]) is based on EMF. We use it to define model transformations that are executed by a transformation engine. In ATL, transformations are defined in modules that consist of declarative transformation rules and helper operations. Using a syntax similar to that of OCL, the transformation rules match model elements in a source model and create elements in a target model. A helper is defined in the context of a metamodel element, to which it effectively adds a feature.

For their input, model transformation tools typically use XMI serialisations of MOF-based (meta)models. In the case of UML, these models can simply be created and visualised using standard UML tooling.

3. Model-Driven Architectural Views

To take advantage of the power of DSMLs for code generation and other automated software engineering tasks and that of UML for documentation, we explicitly distinguish architectural documentation and architectural models. We make this concrete by revisiting the conceptual model of the industry standard for description of software architectures (IEEE Std 1471-2000 [16]). The result, the Model-Driven Architectural Views (MDAV) framework, is displayed in Figure 1.

3.1. MDAV Framework

In Figure 1, for the development of a software System, an Architecture is defined that includes the most important design decisions. These are made concrete in an Architectural Description that consists of Models on the one hand, and architectural Views on the other. In the spirit of MDE, models conform to a Metamodel and are used for several (automated) tasks such as, analysis and code generation. Views on the other hand conform to a Viewpoint and are primarily used for communication purposes. Both metamodels and viewpoints are developed to address a certain set of Concerns. A viewpoint prescribes the language to be used to model the architecture. A metamodel specifies the abstract syntax of this language.

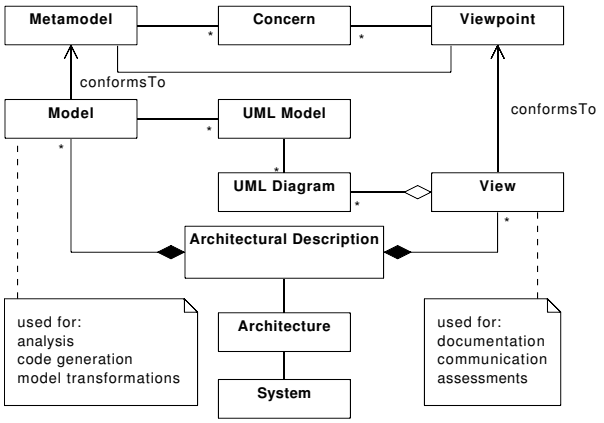


Figure 1. MDAV framework

A view includes diagrams in its primary presentation that represent the associated architectural models.

To allow the use of custom defined DSMLs without the need to specifically develop corresponding graphical representations and editors, we use UML Diagrams. To this end, we map DSML Models to UML Models that are visualised as UML diagrams for inclusion in view documentation with standard UML tooling. Thus, in MDAV the connection between views and models is made through (UML) diagrams. Thanks to this connection, views become model driven.

3.2. MDAV Process

In summary, compared to the conceptual model as described by IEEE Std 1471-2000, we add the concept of a diagram that allows to relate a view to a model. Furthermore, in-line with MDE, we explicitly added a metamodel as a description of the modelling language used in a view. Application of the corresponding approach involves three steps: definition of 1) a suitable metamodel, 2) means to create corresponding models, and 3) a mapping to UML.

A suitable metamodel for a particular viewpoint can be defined from scratch or based on an existing ADL that addresses the relevant concern. In the former case, we use a description of the viewpoint (e.g., from Clements et al. [15]) and create corresponding elements and relations in the metamodel. In the latter case, we base the metamodel on the ADL's grammar (or other language specification mechanism). Given the typically modest size and simple syntax of ADLs and using appropriate tooling, corresponding metamodels are easily created.

A means to create models associated with the defined metamodel is also required. Depending on the complexity of the associated metamodel different alternatives are suitable, of which we give examples in Section 4.

We specify and implement the mapping between the

Proceedings of the 14th Annual IEEE International Conference and
Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

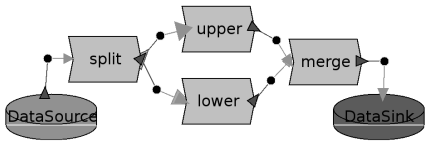


Figure 2. C&C model of CaPiTaLiZe (ACME)

prescribed metamodel and UML using a model transformation language. For several ADLs mappings to UML already exist, that we can specify as model transformations. This allows us to automatically transform an architectural (ADL) model to a UML Model. As such, ADL Models and UML Diagrams can evolve simultaneously.

Although the corresponding UML diagram might not exactly represent the architectural model (e.g., because the latter uses concepts that do not correspond to any UML concept), it is typically complete enough for most communication purposes. Moreover, in the case of a semantic mismatch, we use stereotypes to indicate the type of ADL element a specific UML element represents.

4. Using MDAV to Generate Views

We applied MDAV to two architectural viewpoints: we defined an appropriate metamodel, that is, a modelling 'language', means to create and manipulate associated models, and a mapping to UML,

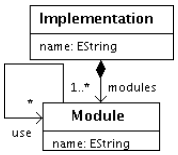
We use the CaPiTaLiZe system [19] as a running example. CaPiTaLiZe transforms a character stream by capitalising alternate characters. A C&C model of CaPiTaLiZe defined using an ADL (ACME [20]) is visualised in Figure 2. CaPiTaLiZe is designed as a pipe-and-filter system, with separate components for splitting a stream of characters in two streams (split), (un)capitalising characters (upper, lower), and merging two streams of characters (merge).

The diagram of CaPiTaLiZe's module view is depicted in Figure 3(c). In this UML class diagram we represent architectural modules with UML Packages and use-relations with UML Dependencies, as suggested by Clements et al. [15].

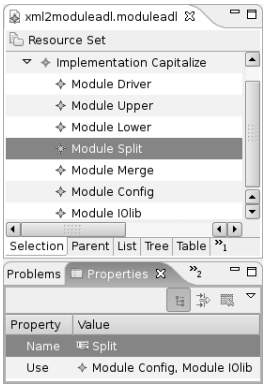
4.1. Module-uses view

Metamodel Module-uses views are based on a special type of dependency relation: the uses relation. As such, these views only contain one type of element and one type of relation [15].

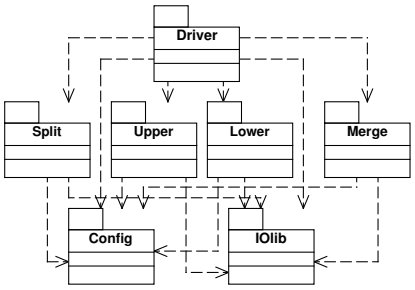
Although UML is well-suited and therefore also typically used in the primary presentation of module views, we developed a small custom metamodel to illustrate MDAV. This MODULEADL metamodel is depicted in Figure 3(a). In addition to a Module element and use relation it defines an Implementation to consist of a set of modules that may use other modules.



(a) MODULEADL metamodel



(b) MODULEADL model of CaPiTaLiZe in EMF editor



(c) MODULEADL diagram (UML)

Figure 3. MODULEADL

Model creation Using MOF, in principle, only the abstract syntax is defined. Although XMI offers an off-the-shelf mapping to XML, it is not intended to be used directly by software developers [21].

For simple metamodels, such as our MODULEADL, we propose to use the editor generated by EMF for the creation and inspection of models. Figure 3(b) shows a screenshot of this editor while editing the MODULEADL model for the CaPiTaLiZe system. The top part shows the modules that are defined for this system, the Properties pane is used to inspect the properties of those modules. This screenshot shows, for instance, that Module Split uses Module Config and Module IOlib.

UML mapping The mapping to UML is based on one of the mappings suggested in [15]. We map Modules to UML Packages and the uses relation to UML Dependencies. We specified this mapping using ATL. A fragment is depicted in Listing 1.

In an ATL transformation rule a **from** clause specifies a pattern that is matched by elements of the source model. For each match the target patterns in the **to** clause are instantiated in the target model. In this case, the Package rule creates a Package (p) and a set of Dependencies (ds) for each Module (m) in the source model. Using the **distinct** ... **foreach** construct a Dependency is created for every Module that is used by the Module that matched the rule (m.use). For both target elements a set of bindings is

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

```
rule Package {
  from m:MADL!Module
  to p:UML!Package (
    name <− m.name,
    clientDependency <− ds),
  ds: distinct UML!Dependency foreach (um in m.use)(
    client <− m,
    supplier <− um)
}
```

Listing 1. MODULEADL → UML (ATL)

specified to initialise their features. The clientDependency feature of the created Package (p), for instance, is initialised with the set of Dependencies created by this rule as well (ds).

The result of applying this transformation to the MODULEADL model of the CaPiTaLiZe system (Fig. 3(b)), is visualised using a UML tool (Fig. 3(c)).

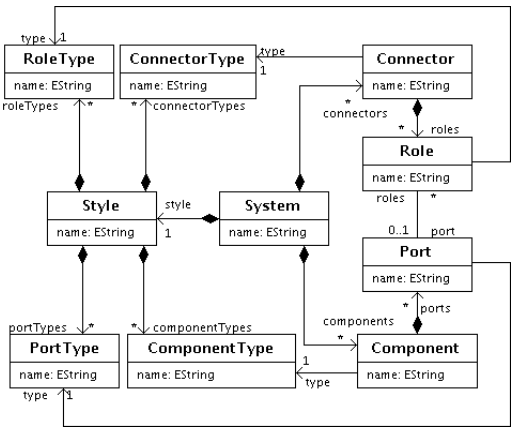
4.2. Component-and-connector view

Metamodel For C&C views, we define a metamodel for a simple ADL similar to ACME [20], an ADL interchange language that covers the most constructs in a wide range of ADLs.

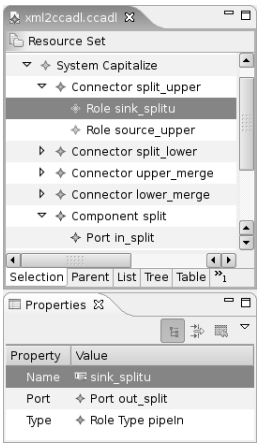
Consider the metamodel for the ADL (CCADL) in Figure 4(a). Using CCADL the architecture of a System consists of a Style, a set of Components, and a set of Connectors. A component owns a set of Ports via which it interacts with its environment. Similarly a connector owns a set of Roles that define what behaviour is expected from the participants in the interaction the connector represents. By attaching conforming roles and ports, configurations of components and connectors can be created. Finally, the style defines the types of components (ComponentType), connectors (ConnectorType), roles (RoleType), and ports (PortType).

Model creation Again, a straightforward approach to create CCADL models is to use the editor generated by EMF. Figure 4(b) displays a screenshot of this editor, while editing the CaPiTaLiZe CCADL model. When considering the complexity of the CCADL metamodel (compared to the MODULEADL metamodel), it becomes clear that editing models using this editor is inconvenient. Using this editor it is not possible, for instance, to immediately determine the component and connector types and understand their configuration.

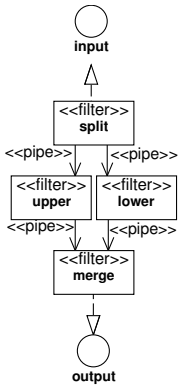
As an alternative, we propose to use a simple XML DTD or schema that allows to describe associated models as simple as possible. A fragment of an XML document conforming to such a DTD describing the same CaPiTaLiZe system is depicted in Listing 2. Note that, the DTD



(a) CCADL metamodel



(b) CCADL model of CaPiTaLiZe in EMF editor



(c) CCADL diagram (UML)

Figure 4. CCADL

we defined allows to separately specify the configuration of components and connectors as a set of attachments.

If we use simple XML documents to specify systems in CCADL, we separately need to populate a *model* conforming to the CCADL metamodel. Several approaches can be used to populate a model.

One possibility is to develop a so-called injector, a program that parses a file and uses the API generated by EMF to instantiate a corresponding model. In general, an injector is used to bridge two different domains, in this case the XML and modelware (MOF) domains. In the context of model-driven engineering such domains are also referred to as Technological Spaces [22].

As an alternative we reuse the XML injector, and XML metamodel (Fig. 5) provided by the ATL project[1]. Based on an XML document this injector instantiates a model that conforms to the XML metamodel. Subsequently, we transform this model into a model that conforms to the CCADL metamodel using ATL model transformations. The latter approach requires the smallest ef-

[1] http://www.eclipse.org/m2m/atl

```
...
<System name="Capitalize">
  <Style name="pf">
    <ComponentType name="Filter"/>
    <PortType name="filterOut"/>
    ...
  </Style>
  <Component name="split" type="Filter">
    <Port name="in_split" type="filterIn"/>
    <Port name="out_split" type="filterOut"/>
  </Component>
  ...
  <Connector name="split_upper" type="Pipe">
    <Role name="sink_splitu" type="pipeIn"/>
    <Role name="source_upper" type="pipeOut"/>
  </Connector>
  ...
  <Configuration>
    <Attach port="out_split" role="sink_splitu"/>
    ...
  </Configuration>
</System>
```

Listing 2. C&C model of CaPiTaLiZe (XML)

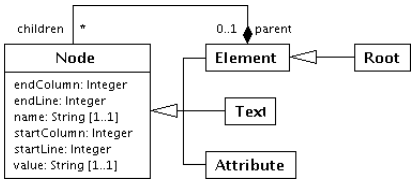


Figure 5. XML metamodel

fort because it reuses existing injectors and metamodels, and only requires us to specify a transformation that maps the XML metamodel to our CCADL metamodel.

The transformation to instantiate a CCADL model based on a (injected) XML source model is straightforward. Listing 3 contains a fragment of this transformation. The rule matches all XML elements named 'Component'. For each it creates a Component in the CCADL model. The type and name features are initialised using two helpers, getType and getName. They navigate the XML model to extract the desired information. For the type feature this is another XML Element that, in turn, matches a rule that creates ComponentTypes. The other elements of the CCADL metamodel are instantiated by similar rules.

UML mapping The UML representation of components and connectors is based on the strategies for modelling architecture with UML described by Garlan et al. [14]. In Figure 4(c) component and connector types are depicted as stereotypes, components as classes, connectors as associations, and ports and roles are not explicitly shown at all.

```
rule Component {
  from el:XML!Element(
    el.name='Component')
  to c:CCADL!Component(
    type <- el.getType,
    ports <- el.children->select(e|e.name='Port'),
    name <- el.getName)
}
```

Listing 3. XML → CCADL (ATL)

```
rule Association {
  from conn:CCADL!Connector
  to asoc:UML!Association(
    name <- '<<' + conn.type.name + '>>',
    connection <-Set{conn.roles->asSequence()->first(),
      conn.roles->asSequence()->last()})
}
rule AssociationEnd {
  from r:CCADL!Role
  to aend:UML!AssociationEnd(
    isNavigable <- r.type.name='pipeOut',
    participant <- CCADL!Component->allInstances()->
     select(c|c.ports->includes(r.port)))
}
```

Listing 4. CCADL → UML (ATL)

Listing 4 shows two rules of the corresponding ATL model transformation. The Association rule instantiates an Association (asoc) for each Connector (conn) in the source model. As our UML tool did not support stereotypes on Associations, we initialise the name feature to mimic one. Although not very elegant, this is acceptable when considering the goal of our transformation: generation of diagrams for documentation. In UML an Association has a connection feature that is a set of AssociationEnds. In our case, these represent the Roles of a Connector. For simplicity we assumed a Connector has exactly two Roles. The connection feature is initialised to the result of the rule that matches the roles of the Connector. Roles are matched by the AssociationEnd rule that creates an AssociationEnd (aend) for every matched Role (r). The isNaviagable feature is initialised depending on whether the matched Role is of type pipeOut (true) or not (false). As such, we control the direction of the Association for representation of the Connector.

Depending on the exact concerns the associated viewpoint addresses, alternative mappings to UML can be implemented similarly, such as a mapping that explicitly shows ports and roles.

5. Industrial Application

In this section we discuss a case study in which we applied MDAV to an architectural view in use for a class of control systems. Before discussing the three steps of our approach, we briefly introduce the case study.

ASML, a large manufacturer of equipment for the semi-conductor industry, studies the migration to a new architecture for supervisory machine control (SMC) components. In an advanced manufacturing machine, such as the wafer-scanners developed by ASML, an SMC component is responsible for the coordination of manufacturing activities in order to perform manufacturing requests. In a layered control architecture, an SMC component receives manufacturing requests from components in a higher layer, and coordinates the execution of manufacturing activities by components in a lower layer. Traditionally, the design for SMC systems is based on state transition models. The new approach [23] is based on task-resource models.

Metamodel Using the task-resource approach, SMC systems consist for a large part of generic, reusable components defined by a product-line architecture. The remaining application-specific components are generated based on a model of an SMC system in terms of tasks and resources. The associated metamodel is shown in Figure 6(a).

Task-resource models consist of a static and a dynamic part. The static part models the controlled System by specifying the Behaviours (manufacturing activities) it can perform, the Capabilities this requires, and the Resources (subsystems) it controls to offer those capabilities. The dynamic part models the manufacturing requests the component can perform in terms of (simple, conditional, or compound) Tasks that are of a specific Behaviour. Precedence relations between tasks are used to specify restrictions on execution order.

Based on the metamodel, tools can be developed for the generation of source code, model validation, and other software engineering tasks that can be automated. We used it, for instance, as the target of a model transformation that automates the migration of SMC components from a state-based to a task-resource-based architecture [6].

Model creation In this case, task-resource models were obtained by the automatic migration of legacy SMC models (based on state machines) to models based on the task-resource architecture. As such, a means to create task-resource models directly was not yet required. When SMC systems are developed based on the task-resource approach from scratch, such means would be required. In that case, one of the alternatives presented in the previous section can be used.

An example of a generated task-resource model (as result of the automated migration) inspected using the EMF editor is depicted in Figure 6(b). This editor was generated based on the metamodel we defined (Fig. 6(a)). Apart from this editor there was no (more advanced) editor available for these models.

UML mapping For the documentation of SMC systems based on the task-resource architecture, a viewpoint was defined. Due to the lack of a convenient editor to visualise task-resource models and to take advantage of available tooling and experience, the viewpoint prescribes that such models are depicted using UML diagrams. As such, the alignment of the task-resource view documentation with the task-resource models from which code can be generated, involved a mapping of the corresponding metamodel to UML.

For the documentation of a conforming view, separate diagrams are used for the static part and for each of the possible requests of the dynamic part. For the former, a UML Class Diagram is used in which a Class with appropriate Stereotype is used to represent a Behaviour, Capability, or Resource. For the latter, UML Activity Diagrams (one for each request) are used that effectively represent tasks and their precedence relations as task graphs. We used ATL to define corresponding mappings from the task-resource metamodel to UML class models for the static part, and to UML activity graphs for the dynamic part. A UML tool visualises these models as a UML Class Diagram, and a UML Activity Diagram, respectively.

As an example, the rule in Listing 5 maps a SimpleTask to the element that represented an activity in a UML Activity Diagram: ActionState. The name feature of the generated ActionState (state) is initialised using the name of the behaviour associated with the SimpleTask (st) that matched the rule. The rule also creates a set of Stereotypes (stype). In that target element the sTypes helper determines the resources required by the behaviour associated with the matched SimpleTask. This set is used to generate a set of ActionState Stereotypes used to initialise the stereotype feature of the generated ActionState.

Application of the transformation we defined to the model partly depicted in Figure 6(b), results in a class model and an activity graph for each request. One of those is visualised as an Activity Diagram in Figure 6(c). Tasks are represented by Activities, required resources by Stereotypes on Activities, precedence relations between Tasks by the order of the Activities, fork bars were used to indicate tasks that can be executed concurrently (i.e., tasks without precedence relations), and conditional Tasks (OrTasks) were mapped to choice nodes. As such, to complete the migration, models as the one depicted

```
rule ActionState {
  from st:TRS!SimpleTask
  to state:UML!ActionState (
    name <− st.behaviour.name,
    stereotype <− stype),
  stype: distinct UML!Stereotype foreach (s in st.sTypes)(
    name <− s,
    baseClass <− 'ActionState'),
  ...
}
```

Listing 5. TRS → UML (ATL)

in Figure 6(b) are used for model-based generation of source code, while diagrams as the one in Figure 6(c) are used for view-based *documentation*. Using model transformations the diagrams for this documentation are generated automatically.

6. Discussion

Our approach has several benefits. It reduces the effort required for the introduction of MDE approaches by circumventing the need to specifically develop graphical editors for the visualisation of DSMLs models. Furthermore it allows to introduce an MDE approach gradually; UML diagrams can continue to be used for documentation purposes. As such, in the case of software architecture, it facilitates the integration of ADLs and supporting tools in industrial development processes.

As presented here the approach uses MDA technology for model transformations and metamodelling. The underlying ideas are applicable to other MDE approaches as well: either by using the available transformation and metamodelling technologies for that MDE approach, or by implementing a bridge to MDA. We gave an example of the latter in Section 4.2 for XML.

Of course, the diagrams that are generated automatically using our approach, only constitute a minor part of the complete documentation. Architectural views, for instance, typically also document (some of) the rationale and trade-offs that underly design decisions [15]. In fact, an architectural view can be seen as 'diagrams + explaining text'. Although the 'explaining text' is not automatically updated using our approach, it does provides a starting point for doing so (i.e., the newly generated diagram).

Whether a mapping to UML is feasible, depends on the type of models involved and the documentation requirements. A potential risk of our use of UML, is that the UML semantics might not match with the semantics of the represented (DSML) model elements, resulting in ambiguities. In these cases appropriate stereotypes should be introduced. As an example, consider the stereotypes in Figure 4(c). These stereotypes are included in the ATL mappings we defined.

In the case that the semantic gap between the involved metamodel and UML is too large to be solved with stereotypes, instead of UML, more generic graph languages such as DOT[2] and GXL[3] could be used as target of the mapping.

The effort required for specification of the mappings to UML is mainly determined by the complexity and size of the DSML metamodel. Typically, these are relatively small (e.g., compared to UML). Furthermore, such mappings can be either specifically developed (as in the case of task-resource models) or reused (as in the case of ADLs). In the latter case they only need to be specified as a model transformation.

Our approach focusses on *visualisation* of DSML models. It does not offer visual *editing* for models conforming to complex metamodels. When that is required, editors have to be developed specifically. Technology to partly generate such editors is provided in the Eclipse Graphical Modeling Framework (GMF [24]) using EMF. Based on the specification of a concrete syntax and the abstract syntax specified by a metamodel this plugin can generate an editor. However, in the case that only visualisation is required, our approach offers a lightweight alternative.

Another alternative is to simply manually create documentation instead of automatically as in our approach. In that case the diagrams corresponding to some software models are created (drawn) manually using modelling or more generic tools. Obviously, consistency becomes an issue with such an approach.

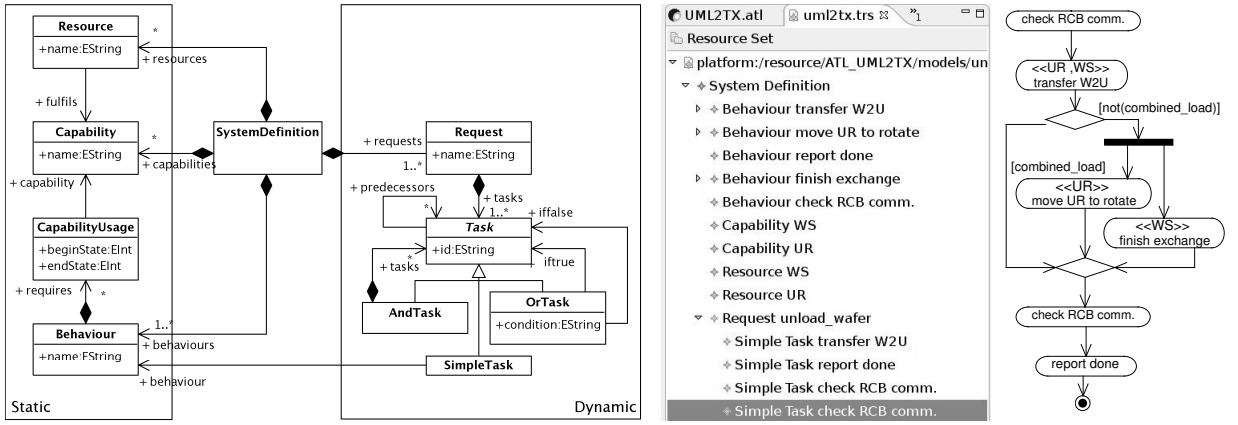
7. Related Work

Fondement and Baar [25] present an approach to specify (graphical) concrete syntax by extending metamodels. Based on this approach tools could be developed to (partly) generate corresponding editors. Instead, we take advantage of existing UML tools.

Medvidovic et al. [12] investigate the use of UML in the domain of software architecture. More in particular, they investigate how modelling constructs used in ADLs (i.e., a type of DSML) can be represented using UML. They consider two approaches for using UML to model software architectures: (1) use UML 'as-is', and (2) use UML's extension mechanisms. They conclude that UML has a number of limitations when used to model software architectures. The lack of architectural modelling constructs makes it necessary to adopt specific interpre-

[2] DOT - Language used by Graphviz (Graph Visualisation Software), see http://www.graphviz.org

[3] GXL - Graph eXchange Language, see http://www.gupro.de/GXL



(a) Task-resource meta-model (b) Task-resource model (c) Task-resource diagram (UML)

Figure 6. Task-resource metamodel, model, and UML representation

tations of UML model elements or to rely on the Object Constraint Language (OCL) to constraint the use of model constructs. In this paper we investigated a third strategy that is based on the definition of metamodels for ADLs and their mapping to UML using model transformations.

Five strategies for representing architectural structure in UML are described by Garlan et al. [14]. They conclude that there is no single best way to do this. Furthermore, they identify a trade-off between completeness and legibility: strategies that assign different UML model elements for each ADL construct (completeness) tend to be very verbose and hence poorly readable (legibility). One of their recommendations to solve this, is to continue to use ADLs but to provide mappings to object-oriented notations. In the current paper we specified such mappings using model transformations, which makes them automated.

Where we propose to use MOF for the definition of DSMLs, Dashofy et al. [26] use XML for the definition of ADLs. It provides generic high-level XML schemas that can be extended for development of ADLs. They leverage the available tool support for XML. As we use MOF, we leverage available UML and MOF tools as well. This enables, for instance, the specification of transformations on a higher level of abstraction by a model transformation language.

8. Concluding Remarks

In this paper we proposed to combine DSML models and UML diagrams for model-driven software documentation. Where MDE approaches typically aim to use DSML models to automatically create source code, our approach complements MDE with the (partial) creation of documentation.

The main motivation for our approach is the observation that although DSMLs have clear advantages over general-purpose modelling languages, it requires considerable effort to develop graphical editors and representations. In particular, the definition and implementation of their concrete syntax or notation is much more involved than that of their abstract syntax, which is supported by technologies, such as MOF and EMF. This is a problem, as graphical representations of models are an essential part of software documentation.

Our approach uses model transformations to (automatically) map DSML models to UML models. These UML models are easily visualised as UML diagrams using available modelling tools. While the DSML models can be used for code generation and other automated software engineering tasks, these diagrams are used in the documentation. As such, our approach allows to optimise both completeness (by the ADL model) and legibility (by the UML diagram) of architecture descriptions. Furthermore, part of the documentation can be automatically updated as the software system evolves.

Application of our approach requires the definition of a DSML metamodel using MOF and mappings to UML using model transformations. This needs to be done once for each DSML used. Furthermore, a means to create associated models is required. We gave several examples for this. Compared to the development of a complete graphical editor for the defined metamodel, our approach is more lightweight.

We evaluated our approach in the domain of software architecture, for which we defined MDAV. It refines the industry standard for architecture documentation (IEEE Std 1471-2000) by linking architectural views (documentation) to architectural models using model

transformations and UML. MDAV is easily generalised to other domains. As an example, we discussed an industrial application in the domain of control systems.

Currently we are investigating how the proposed model transformations can best be integrated with existing tooling and development processes. Another problem we are investigating is the (automatic) derivation of metamodels (e.g., based on MOF) from grammars (e.g., based on EBNF). A solution to this problem increases the effectiveness of our approach when applied to existing DSMLs that are not based on MDA technology.

Acknowledgement Part of the research described in this paper was sponsored by NWO via the Jacquard Reconstructor project.

References

[1] Jean Bézivin. On the unification power of models. *Software and Systems Modelling*, 4(2):171–188, May 2005.

[2] Nenad Medvidovic and Richard N. Taylor. A framework for classifying and comparing architecture description languages. In *ESEC '97/FSE-5: Proc. 6th European conference held jointly with the 5th ACM SIGSOFT international symposium on Foundations of software engineering*, pages 60–76. Springer-Verlag, 1997.

[3] Duncan Doyle, Hans Geers, Bas Graaf, and Arie van Deursen. Migrating a domain-specific modeling language to MDA technology. In *Proc. 3rd Int'l Workshop on Metamodels, Schemas, Grammars, and Ontologies for Reverse Engineering (ateM 2006)*, 2006.

[4] Arie van Deursen and Paul Klint. Little languages: Little maintenance? *J. Software Maintenance: Research and Practice*, 10(2):75–92, December 1998.

[5] Bas Graaf. Model-driven consistency checking of behavioural specifications. In *Proc. 4th Int'l Workshop on Model-based Methodologies for Pervasive and Embedded Software (MOMPES 2007)*, 2007. Accepted for publication.

[6] Bas Graaf, Sven Weber, and Arie van Deursen. Model-driven migration of supervisory machine control architectures. *J. Systems and Software*, 2007. Accepted for publication.

[7] OMG. Meta-object facility (MOF). `http://www.omg.org/mof`, 2005.

[8] Eclipse Foundation. Eclipse modeling framework (EMF). `http://www.eclipse.org/emf`, 2005.

[9] Christian F.J. Lange, Michel R.V. Chaudron, and Johan Muskens. In practice: UML software architecture and design description. *IEEE Software*, 23(2):40–46, March 2006.

[10] OMG. MDA. `http://www.omg.org/mda`, 2005.

[11] Philipe Kruchten, Henk Obbink, and Judith Stafford. The past, present, and future of software architecture. *IEEE Software*, 23(2):22–30, March 2006.

[12] Nenad Medvidovic, David S. Rosenblum, David F. Redmiles, and Jason E. Robbins. Modeling Software Architectures in the Unified Modeling Language. *ACM Trans. Softw. Eng. Methodol.*, 11(1):2–57, 2002.

[13] Bas Graaf, Marco Lormans, and Hans Toetenel. Embedded software engineering: The state of the practice. *IEEE Software*, 20(6):61–69, November–December 2003.

[14] David Garlan, Shang-Wen Cheng, and Andrew J. Kompanek. Reconciling the needs of architectural description with object-modeling notations. *Science of Computer Programming*, 44:23–49, 2002.

[15] Paul Clements, Felix Bachmann, Len Bass, David Garlan, James Ivers, Reed Little, Robert Nord, and Judith Stafford. *Documenting Software Architectures:Views and Beyond*. Addison-Wesley, 2002.

[16] IEEE-1471. IEEE recommended practice for architectural description of software intensive systems. IEEE Std 1471–2000, 2000.

[17] Object Management Group. XML metadata interchange formal specification. `http://www.omg.org/technology/documents/formal/xmi.htm`, January 2002–03.

[18] Frédéric Jouault and Ivan Kurtev. Transforming models with ATL. In *Proc. Model Transformations in Practice Workshop at MoDELS2005*, 2005.

[19] Robert Allen and David Garlan. A formal basis for architectural connection. *ACM Trans. Software Engineering and Methodology (TOSEM)*, 6(3):213–249, July 1997.

[20] David Garlan, Robert T. Monroe, and David Wile. ACME: architectural description of component-based systems. In *Foundations of component-based systems*, pages 47–67. Cambridge University Press, 2000.

[21] Timothy J. Grose, Gary C. Doney, and PhD. Stephan A.Brodsky. *Mastering XMI. Java programming with XMI, XML, and UML*. OMG Press. Wiley, 2002.

[22] Ivan Kurtev, Jean Bézivin, and Mehmet Aksit. Technological spaces: an initial appraisal. In *Confederated Int'l Conferences CoopIS, DOA, and ODBASE 2002*. Springer-Verlag, 2002. Industrial Track.

[23] N.J.M. van den Nieuwelaar. *Supervisory Machine Control by Predictive-Reactive Scheduling*. PhD thesis, Technische Univ. Eindhoven, 2004.

[24] Eclipse Foundation. Eclipse graphical modeling framework (GMF). `http://www.eclipse.org/gmf`, 2006.

[25] Frédéric Fondement and Thomas Baar. Making metamodels aware of concrete syntax. In *Proc. 1st European Conf. Model Driven Architecture - Foundations and Applications (ECMDA-FA 2005)*, volume 3748 of *Lecture Notes in Computer Science*, pages 190–204. Springer-Verlag, 2005.

[26] Eric M. Dashofy, André van der Hoek, and Richard N. Taylor. A comprehensive approach for the development of modular software architecture description languages. *ACM Trans. Software Engineering and Methodology*, 14 (2):199–245, April 2005.

State-Based Modeling to Support the Evolution and Maintenance of Safety-Critical Software Product Lines

Jing Liu[1], Josh Dehlinger[1], Hongyu Sun[1] and Robyn Lutz[1, 2]
[1]*Department of Computer Science, Iowa State University*
[2]*Jet Propulsion Laboratory/Caltech*
{janetlj, dehlinge, sun, rlutz}@cs.iastate.edu

Abstract

Changes to safety-critical product lines can jeopardize the safety properties that they must ensure. Thus, evolving software product lines must consider the impact that changes to requirements may have on the existing systems and their safety. The contribution of this work is a systematic, tool-supported technique to support safe evolution of product-line requirements using a model-based approach. We show how the potential feature interactions that need to be modeled are scoped and identified with the aid of product-line software fault tree analysis. Further, we show how reuse of the state-based models is effectively exploited in the evolution phase of product-line engineering. To illustrate this approach, we apply our technique to the evolution of a safety-critical cardiac pacemaker product line.

1. Introduction

Changes to software requirements after deployment, due to system evolution, increase the difficulty of understanding, tracing, modeling and verifying the effects on system safety properties and can jeopardize the safety of the system [8]. Changes to the software requirements of a product line can greatly increase this difficulty because multiple systems are involved that may have varying safety properties and that can jeopardize the safety of the systems in different ways [6]. Safety-critical product lines, including cardiac pacemakers [20] [21], constellations of satellites [9] and medical imaging systems [28], need techniques and tools to accommodate and analyze the impact of system evolution on the product line and the product line's safety properties [17].

A product-line is a set of systems developed by a single company that share a common set of core requirements (i.e., the product line's commonalities) but differ amongst each other according to a set of managed variable requirements (i.e., the product line's variabilities) [27]. The utilization of product-line engineering for software systems is advantageous in that it exploits the reuse potential in the analysis, design and development of the commonalities and variabilities in each product-line member [30]. Studies suggest that product-line engineering can reduce development time and cost as well as increase the quality of products by a factor of 10 times or more [6]. Product-line evolution typically involves the addition of new features (i.e., variabilities) or the refining of existing variabilities (i.e., altering the allowed parameters of a product-line variability) [29].

Yet, product-line engineering still lacks the technical mechanisms to efficiently ensure the safety of each product-line system while fully taking advantage of its reuse potential [23]. Specifically, Kang [17] identifies the following as open problems for the viable use of product-line engineering:

- Verifying quality attributes (e.g., safety and reliability) and detecting feature interactions that may violate the quality attributes
- Modeling, analyzing and managing product-line features and feature interactions while avoiding the feature explosion problem
- Accommodating the evolution of the product line and adapting the product-line assets to the evolved requirements

The work described here addresses these problems in the context of the evolution and maintenance of a product line. Specifically, this work provides a structured, tool-supported decision mechanism, driven by the use of a product-line software fault tree analysis, to determine if new requirements, as a result of product-line evolution, can be safely integrated into the product line without introducing unchecked safety concerns. We utilize a product-line requirements analysis tool [25] and product-line software fault tree analysis tool [10] to augment and focus our state-based modeling of a product line on those new requirements

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

and potential feature interactions that may be safety-critical.

The contributions of this work are a tool-supported, state-based, safety analysis approach for the evolution of a software product line, including:

- Linking safety-critical, product-line requirements to their state-based model components
- Identifying and analyzing potential safety-critical feature interactions
- Modifying and reusing existing product-line state-based models to include new requirements from product-line evolution

This work is a part of a larger effort that investigates how safety-critical product lines evolve and that develops analysis techniques, tools and strategies to reduce the cost of safety analysis and enhance the safety and reusability of evolving product lines. The long-term goal is to provide safety analysis results for the new systems of a product line during requirements evolution in a timely and cost-efficient manner.

The remainder of this paper is as follows. Section 2 reviews related research in product-line engineering, state-based modeling for product lines and product-line safety analysis. Section 3 gives an overview of our approach to accommodate the safety analysis of evolving software product lines using state-based modeling. Section 4 details our technique using the evolution of a pacemaker as our safety-critical case study. Section 5 provides a brief discussion of our technique and our experience in its application. Finally, Section 6 provides some concluding remarks.

2. Background & Related Work

This work builds upon previous work integrating product-line engineering, state-based modeling and software safety analysis. Compared to our previous work in this field [20] [21], this work demonstrates how the safety analysis of a product line using a state-based modeling approach can accommodate product-line evolution.

2.1. Software Product-Line Engineering

The ability to reuse software engineering assets during system development continues to be of vital interest to industry as it offers the possibility to significantly decrease both the time and cost of software requirements specification, development, maintenance and evolution [27]. In product-line engineering, the common, managed set of features shared by all members, the commonalities, are reused for all members of the product line. For example, a commonality for a pacemaker is "*A pacemaker's pacing cycle length shall be the sum of the senseTime and the refractoryTime*".

The variabilities of a product line differentiate the product-line members and may have a design, configuration, delivery or run-time binding with the product-line member [20] [21]. For example, a run-time binding pacemaker variability is "*The senseTime of a pacemaker's pacing cycle may vary at run-time by setting the senseTime from 800 msec to 300 msec*".

Product-line dependencies restrict which combinations of variability subsets can form viable product line members. Dependencies may enforce safety requirements by preventing or restricting some feature interactions. For example, a pacemaker dependency is "*A modeTransitive type pacemaker must only use a 800 msec senseTime when it is operating in a Inhibited pacing mode*".

Product-line engineering is typically partitioned into two phases: domain engineering and application engineering [30]. A product line is initially defined by its commonalities and variabilities in the domain engineering phase. The benefits of product-line engineering come in the application engineering phase when the reusable assets defined in the domain engineering phase are exploited to create product-line members. Product-line evolution typically involves the addition of new features (i.e., variabilities) or the refining of existing variabilities (i.e., altering the allowed parameters of a product-line variability) [29]. For instance, a requirement evolution for the pacemaker variability given above may expand allowable senseTime pacing cycle to also include some value between 800 msec and 300 msec, e.g., 500 msec.

2.2. Model-Based Software Development

State-based modeling has previously been used as a mechanism to detect the correctness of the requirements and design as well as to aid in the verification of behavioral requirements [1], [7]. Harel and Marelly, like us, have used a scenario-guided approach to testing state-based models as a way to identify missing requirements [16]. However, their work concentrates on validating the safe behavior of single systems, whereas the work described here aims at validating the safe behavior of the multiple systems within a product line.

Software product lines have been modeled in various ways using extensions of UML to aid in the reuse of UML assets. For example, Clauss extends UML to support features diagrams as well as extending the package diagram to incorporate variabilities descriptions [5]; Doerr classifies the relationships within a variation model and relates them to UML

notation [12]; Gomaa uses executable UML statecharts to as a product-line model [15]; and Prehofer uses state-model composition to evaluate the interaction of features [26]. The work described in this paper also uses executable UML but focuses on providing assurance to the satisfaction of the safety properties of the product line as well as examining the potentially unsafe feature interactions.

More recently, Deng, Lenz and Schmidt have demonstrated a model-transformation approach using the Domain Specific Modeling Language to address the changes in a product line's architecture as a result of domain evolution [11]. Our work concentrates on the impact of software evolution on the safety of the system, rather than on the architectural impact.

2.3. Software Safety Analysis

Safety analysis for software product lines is still immature. Safety analysis approaches have been proposed to verify safety properties and discover missing safety requirements for the multiple systems of a product line. Feng and Lutz [14] propose a bi-directional approach that uses a forward search to discover the effects of a hazard coupled with a backward search from faults to their contributing causes to verify and discover safety requirements. Lu and Lutz propose a failure contribution analysis for product lines to help the analysis of the contributions of commonality and variability trees to root node hazards [22]. Yet, these two approaches rely on a static analysis of the product-line requirements rather than the executable analysis done in this work.

This work utilizes two tool-supported product-line safety analysis methods to support the creation of state-based models and to analyze the evolution and feature interactions of product-line requirements.

DECIMAL is a product-line requirements analysis tool that documents the commonalities, variabilities and dependencies of a product line during the domain engineering phase [25]. During the application engineering phase, DECIMAL verifies that the selection of variabilities for a product-line member do not violate the product line's prescribed dependencies.

PLFaultCAT is a tool that aids the construction and analysis of product-line software fault tree analyses (SFTA) [10]. A SFTA is a widely used backward safety analysis technique designed to trace the causal events of a specified hazard down to the basic faults of a single system [18]. PLFaultCAT allows engineers to construct the product-line SFTA and associate the commonalities and variabilities, from DECIMAL, with the leaf nodes of the SFTA in the domain engineering phase. During application engineering, PLFaultCAT semi-automatically produces the product-line members' SFTAs from the product-line SFTA.

The work reported here, as in our previous work [20] [21], uses executable UML within the Rhapsody software modeling environment as well as the TestConductor tool by I-Logix [24].

3. Approach

This section describes the construction of the safety analysis of an evolving software product line using state-based modeling. It focuses on how to identify, model and analyze potentially unsafe feature interactions.

3.1. Safety Analysis of Evolving Software Product Lines Using State-Based Modeling

We here provide a step-by-step overview of our technique for safety analysis of software product lines using state-based modeling for a product line during evolution.

3.1.1. Commonality and Variability Analysis. The Commonality and Variability Analysis (CVA) documents the product line's requirements [30]. During evolution, new feature requirements (i.e., variabilities) are added to the CVA, possibly using a product-line requirements analysis tool, such as DECIMAL [25], as done here.

3.1.2. Product-Line Software Fault Tree Analysis (SFTA). A product-line SFTA will need to accommodate the new features if they can potentially contribute to causing one of the failures described in the SFTAs. The new features may require the modification of the product-line SFTA by adding entirely new fault trees as a result of the possibility of new root node hazards occurring. This requires the construction of a product-line SFTA just as done during the initial development of a product line [10].

Additionally, new features introduced during product-line evolution may need to be included in existing product-line SFTAs. To accomplish this, each existing fault tree is analyzed to see how the new feature(s) can contribute to cause the root node hazards. This may entail adding subtrees to the existing fault trees or associating the requirements of the new feature with the leaf nodes of the fault tree. Here we use the SFTA tool PLFaultCAT [10], to achieve this.

3.1.3. Variation model generation. We map the leaf nodes of the product-line SFTA to architectural components and then model the behavior of the

architectural component in a state model. During the initial development of a product line, the state-chart model is incrementally built from the product that has the fewest variable features until all features are included into the state model [20] [21].

To address product-line evolution, any new features are incrementally integrated into the state model. To achieve this, any newly created SFTAs, a result of Step 2, will need to map the SFTA's leaf nodes to a new or existing architectural component. If they are mapped to an existing component, that component's behavior must be modified to include the new behavior introduced by the new feature(s). If they are mapped to a new architectural component, that new component's behavior should then be modeled and integrated into the product-line state model. For the existing product-line SFTAs that were modified to accommodate the new features, we need to include the new behaviors into the architectural components representation in the state model.

3.1.4. Scenario derivation. Using the product-line SFTA, we derive required scenarios (i.e., those scenarios that enforce a safety property) and forbidden scenarios (i.e., those scenarios that emulate a hazard). For the newly created product-line SFTAs, the process described in [20] [21] suffices. For the existing product-line SFTAs that were modified as a result of the new features, the scenarios that were developed during the initial product line's construction must be altered to accommodate the behavior described in the new subtrees of the SFTAs. This will result in modified testable scenarios that need to be re-executed in Step 5 since the behavior they display as a result of evolution may differ from when they were executed and verified during the product line's initial development.

3.1.5. Scenario-guided model analysis. The developed scenarios, from Step 4, are exercised against the state model, from Step 3. Although the introduction of the new feature as a result of evolution may not have altered all the testable scenarios from the initial development, all scenarios should be exercised against the model to ensure that the inclusion of new features' behavior into the model does not produce undesired/unknown effects (i.e., a regression testing approach). This step then follows [20] [21]: a failure in the execution of the required scenarios indicates inconsistencies between the model execution and the specified scenario; a forbidden scenarios execution will indicate a need to update the design if it is found to allow illegal/hazardous behavior. In each case, an update to the design is warranted if undesired behavior is detected when executing the scenarios in the state model. In this work, we used TestConductor to exercise the model in the Rhapsody modeling environment [24].

3.2. Identifying and Modeling Safety-Critical Feature Interactions

The evolution of a software product line is more complex than for single software systems since new, possibly conflicting, features from the existing products in the product line and the newly introduced features for the new products can result in unsafe/undesirable feature interactions [29]. For example, the 1996 explosion of the initial flight of the Ariane 5 rocket was partially blamed on the interaction of the new features introduced in the Ariane 5 with the features retained from the earlier, Ariane 4, rocket [19]. Thus, it is crucial to ensure that product-line evolution does not introduce feature interactions that compromise the safety properties that the product line previously ensured.

To address this, our approach focuses on the identification and modeling of safety-critical feature interactions to determine whether they may cause a hazard. In the case that the feature interactions could cause a hazard, we explore in simulation the effects of possible alternatives in the model to prevent such an unsafe feature interaction. The identification and analysis of new safety-critical feature interactions in the product line introduced as a result of the inclusion of a new feature(s) during evolution consists of the steps described below.

3.2.1. Identification of safety-critical feature interactions. A product-line software fault tree analysis (SFTA) associates a product line's requirements (i.e., commonalities and variabilities) with the leaf node failure events that may lead to the occurrence of the root node hazard. As described in Section 3.1, Step 2 as well as in [10], the evolution of a product line will associate new product-line requirements with the leaf nodes of existing SFTAs along with former requirements.

After the adaptation of the product-line SFTAs to the new features introduced as a result of evolution, the safety-critical feature interactions can be identified by searching for those product-line requirements that frequently contribute to the possible causes of the fault tree's failure nodes. PLFaultCAT [10], the product-line SFTA tool used here, can automatically identify those product-line requirements and combination of product-line variabilities (i.e., features) that contribute to the most potential failures as defined in the SFTA.

This analysis provides a prioritized list of those product-line requirements and feature interactions that warrant further scrutiny using an executable state-based model. That is, those product-line requirements and feature interactions that are deemed to contribute to the most fault tree failure nodes are more likely to have unsafe interactions with existing product-line requirements and should have their behaviors modeled in order to determine the safe/unsafe behaviors using a dynamic analysis.

3.2.2. State-based modeling of feature interactions to determine safe/unsafe behavior. We here describe our approach using executable state models and scenarios. To determine the safety of feature interactions using our state-based model, we first take a manageable sub-tree of the product-line SFTA. The variabilities in the cut-set of such a fault tree can be used to map to components in the architecture diagram of the product line. If there are new features introduced into the product line, we need to update the architecture design if such a new feature will introduce new components or new associations between components.

Next, we take the state models of those components where the safety related variabilities reside. For newly introduced features, it is likely that the existing state models will be updated, or new state models will be created. We then derive the events that are identified as potentially causing hazards and their direct consequences from SFTA. The causative events and their consequences, represented as message passing between involved components, form required or forbidden scenarios to analyze in the following steps.

Next, we execute the models and inspecting the execution sequences either manually or automatically, with the scenarios previously identified as guidance. The manual inspection includes monitoring the message-passing sequence diagram among the components that are identified in the derived scenarios, pausing at points of importance, and selectively investigating details in the animated statechart view of a specific component when necessary. Manual inspection also includes injecting events at run-time to test the response of the system under different environmental inputs.

The automatic inspection involves using a scenario-based state model testing tool, such as TestConductor, that captures requirements regarding absolute or partial ordering of messages as sequence diagrams, and testing the sequence diagram against the actual order of message-passing during the state model execution. Such tests can be automatically executed for improved inspection.

The manual inspection is more flexible and more informative for requirements that cannot be easily modeled by sequence diagrams due to the nature of the requirements or limits of the tool [20] [21], while the automatic inspection is more thorough, providing more assurance regarding a testable scenario. The outcome gives users information regarding whether a forbidden scenario is likely to happen and how it may happen, or whether a required scenario can sometimes not happen. This is because the execution will give the actual execution scenarios providing details confirming or contradicting the scenarios derived. Note that this scenario-guided inspection of model execution gives no guarantee as to whether a required scenario is always going to happen or a forbidden scenario is never going to happen – that requires the more rigorous reasoning provided by formal methods [4].

After inspection, we need to find mechanisms to avoid forbidden scenarios from happening or enforcing required scenarios, using the detailed results from the previous step as guidance. Such mechanisms, once implemented in the state model, will again be inspected during execution, as described above, to decide if they do achieve their goals. Once confirmed, these mechanisms can be used to suggest new requirements update.

While a new feature is likely to update an existing SFTA or even introduce a new SFTA, previous SFTA-related state models may be re-validated (by running through the process described above) to ensure that the new feature does not interfere with them. Such a re-validation process can be done by adding the new feature related component into the scenario to check if there is any potential interaction between this component and the components residing in the original sequence diagram.

4. Application to the Evolution of a Product-Line Pacemaker

This section applies the approach described in Section 3 to a safety-critical, product-line cardiac pacemaker.

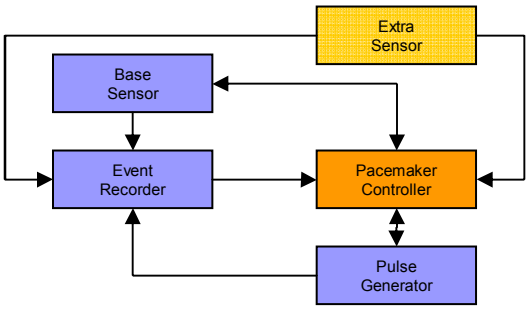


Figure 1. Product-Line Architecture after Evolution

4.1. Description and Evolution

To illustrate our approach, we build upon the pacemaker product line described in [20] [21]. A pacemaker is an embedded medical device designed to monitor and regulate the beating of the heart when it is not beating at a normal rate. It consists of a monitoring device embedded in the chest area as well as a set of pacing leads (wires) from the monitoring device into the chambers of the heart [13]. In our simplified example, the monitoring device has three basic components: a sensing component (sensor) that senses heart beat, a stimulation component (pulse generator) that generates pulses to the heart, and a controlling component (controller) that configures different pacing and sensing algorithms and issues commands. Here, we only consider a single-chambered product line of pacemakers that does pacing and sensing in the heart's ventricles.

Our simplified pacemaker product line consists of the following products and features:

- **BasePacemaker** – This product has the basic functionality shared by all pacemakers: generating a pulse whenever no heart beat is sensed during the sensing interval.
- **ModeTransitivePacemaker** – This product can switch between InhibitedMode and TriggeredMode during runtime. In the InhibitedMode, the pacemaker acts exactly like a BasePacemaker. In the TriggeredMode, a pulse follows every heartbeat to provide a different type of therapy.
- **RateResponsivePacemaker** – This product acts similarly to the BasePacemaker but contains an extra sensor allowing it to adjust its sensing interval according to the patient's current activity level: LRLrate, for a patient's normal activities and URL rate, for when a patient is exercising.
- **ModeTransitive-RateResponsivePacemaker** - This product combines the features of the ModeTransitivePackemaker and the RateResponsivePacemaker.

The evolution of the pacemaker product line that we consider here involves the addition of an EventRecorder component, shown in Fig. 1, to log critical events in the major components of a pacemaker and is used for making therapy decisions. For instance, EventRecorder calculates the number of heart beats sensed by BaseSensor during a fixed recording interval and compares that value with some threshold value to decide if the pacemaker should switch between InhibitedMode and TriggeredMode during run-time. Different pacemakers can log different events at different times, as shown in Table 1.

The addition of the EventRecorder feature was included into the product line's requirements using DECIMAL [25].

Due to the cross-cutting nature of the EventRecording feature, the risk of unsafe feature interaction is higher. For example, when the average number of heart beats in a 6000 msec recording interval exceeds a 24-beat threshold, the EventRecorder shall consider that the patient's heart is fibrillating, so it will command the PacemakerController to switch from InhibitedMode to TriggeredMode to defibrillate it. It must be ensured that the features added to PacemakerController due to the introduction of EventRecorder interact with the existing features in PacemakerController in a predictable manner and that there are no unexpected and/or unsafe feature interactions.

Table 1. Event Recording Feature's Commonality and Variability

Product Name	Component Name	Events to Log
Base Pacemaker	Base Sensor	Average heart rate sensed every fixed recording interval
	Pulse Generator	The pulse width of every pulse being made
Mode Transitive Pacemaker	Base Sensor	Average heart rate sensed every fixed recording interval
	Pulse Generator	1) In the Triggered mode, the average number of pulses generated every fixed recording interval 2) In the Inhibited mode, the pulse width of every pulse being generated
Rate Responsive Pacemaker	Base Sensor	Average heart rate sensed every fixed recording interval
	Pulse Generator	The pulse width of every pulse being made
	Extra Sensor	The percentage of the pacemaker sensing at LRLrate every fixed recording interval

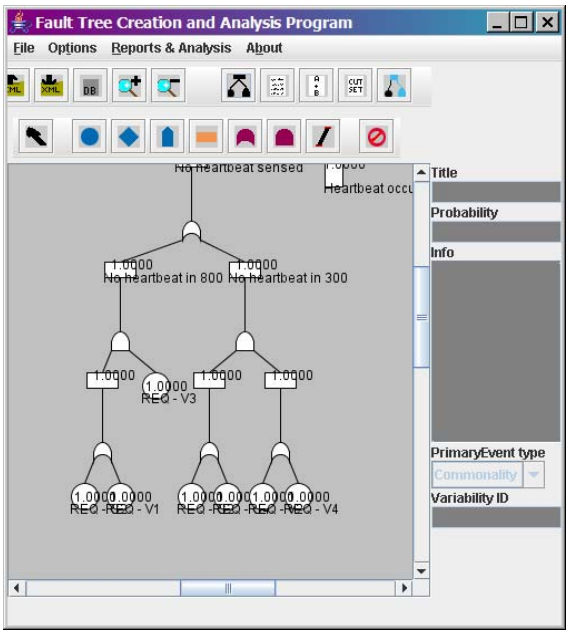


Figure 2. Excerpt of Pacemaker Product-Line SFTA in PLFaultCAT after Evolution

4.2. Product-Line SFTA Evolution

The inclusion of the EventRecorder feature into the pacemaker product line required both the updating of existing product-line software fault tree analyses (SFTA) and the creation of new product-line SFTAs to accommodate the new failure modes that the new feature brings to the product line. For example, because of the new behavior introduced by the EventRecorder feature, a product-line SFTA with a root node of "Failure to switch modes", shown in Fig. 2, had to be added. The creation of the new product-line SFTA with a root node of "Failure to switch modes" required the association of the requirements of the new EventRecorder feature as well as those features from existing product-line products. For example, as illustrated in Fig. 3, the ModeTransitive feature (found in the ModeTransitivePacemaker and the ModeTransitive-RateResponsivePacemaker products) may interact with the EventRecorder feature to cause a hazard. Yet, from examining the SFTA, it is not entirely clear how these two features can interact to cause such hazards, thus the need for further analysis.

Using PLFaultCAT, we can analyze the set of product-line SFTAs to find other such combinations of features that may cause hazards to direct the safety analysis, described in Section 4.3, to those feature interactions, like shown in Fig. 3, which may need to be further scrutinized.

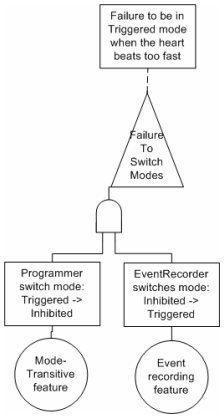


Figure 3. Excerpt of Product-Line SFTA Illustrating Potential Feature Interactions

4.3. State-Based Modeling of Safety-Critical Feature Interactions and Derivation of New Safety Requirements

This section illustrates the steps involved in using the state-based modeling approach to promote safe evolution of a product line. It uses the example of an EventRecording feature introduced as an existing pacemaker product line evolved.

The new EventRecording feature has introduced a possible hazard, "failure to be in the TriggeredMode when the heart beats too fast", shown in Fig. 3. The subtree shown here concretizes this high-level hazard by adding events (e.g., the mode switch event sent from the operator and the mode switch event sent from the EventRecorder), conditions (e.g., the heart beats too fast), and the consequences (both safe and unsafe, e.g., required scenario: remains in TriggeredMode; forbidden scenario: fails to remain in TriggeredMode). The refinement of the hazard node in this way forms the scenario to check against the state models.

For the subtree in Fig. 3, we model the components that implement the leaf node requirements. For example, the ModeTransitive feature is implemented by the PacemakerController component, the MotionSimulator component, and the ExtraSensor component.

After the state models are generated, we instantiate the scenario captured in the subtree by mapping the events, conditions, and consequences to model-level elements. For example, the mode switch event sent from the operator is mapped to the "evInhibitedMode()" message, and the mode switch event sent from the EventRecorder is mapped to the "evTriggeredMode()" message, while the "heart beats too fast" condition is represented by a concrete threshold for the heart beats (16 beats while in

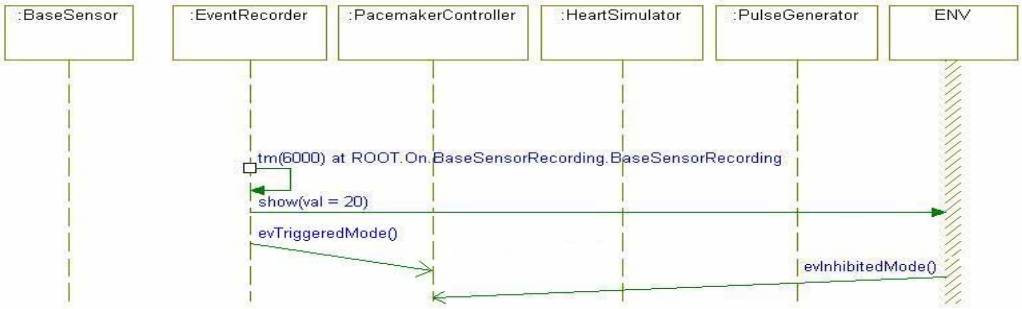


Figure 4. Excerpt of Pacemaker Animated Sequence Diagram

LRLrate, and 24 beats while in URLrate). The safe and unsafe consequences are mapped to PacemakerController component being in the TriggeredMode state and the InhibitedMode state, respectively.

Note that simple state models of other components, even if they do not directly implement the leaf node requirements, such as the PulseGenerator component, can also be generated if their responses in the execution help illustrate the above scenarios more clearly.

The next step animates the generated state models using Rhapsody. The animation process, as explained in Section 3.2, is mainly composed of animated sequence diagrams illustrating message passing during model execution, and animated state charts illustrating states and transitions taken at run time.

Fig. 4 shows a portion of the animated sequence diagram. It is a point where the fixed recording interval has expired (as shown by the "tm(6000)" message), Since the recorded number of heart beats is 20 (greater than the 16 threshold, indicating that the heart is beating too fast), the EventRecorder commands PacemakerController to switch mode from InhibitedMode to TriggeredMode. However, if we then inject an evInhibitedMode() event to PacemakerController, it will switch back to Inhibited mode, as shown in the animated statechart in Fig. 5 (the current states are highlighted), *despite the fact that the heart is still beating too fast.*

The animation shows that the scenario we captured in the fault tree, Fig. 3, and instantiated in the model level can actually happen. It also shows how this unsafe scenario can happen: when the two events (evInhibitedMode() and evTriggeredMode()) occur in a certain order (evTriggeredMode() first and evInhibitedMode()) second). As we discuss below, it is these sorts of ordering and timing issues that the executable state model, unlike the fault tree, can reveal.

Since the animation is done with the executable models, it also provides concrete insights into how to mitigate this potentially hazardous scenario, namely by adding a mechanism that locks the PacemakerController in Triggered mode when the heart beats too fast. The benefit of model-level analysis is that the new mechanism can be tested right away to see if it conforms to the safe scenario.

Another benefit is that we can readily investigate several mechanisms in order to select the more reliable and easy-to-implement one. For example, there are at least two ways that we can implement the mitigation mechanism. One option is to name the mode-switch messages sent from the EventRecorder and the Programmer differently and to give the EventRecorder's message higher priority than the Programmer's. Another option is to set up an internal variable in the PacemakerController that records the heart's status as beating too fast or not. Such a variable is used for guarded transitions from the TriggeredMode state to the InhibitedMode state and can only be changed when the EventRecorder detects a heartbeat.

While both mechanisms prevent the unsafe scenario from happening, the first one is more restrictive in that it grants the EventRecorder priority on messages switching both into, and out of, TriggeredMode The second one just enforces the EventRecorder's priority in switching out of TriggeredMode. However, the second alternative allows the possibility of the Programmer and the EventRecorder racing to switch into TriggeredMode. If the second alternative is selected, this suggests that we may want to introduce additional requirements to handle this possible race condition.

5. Discussion

This work utilized a product-line software fault tree analysis (SFTA) and state-based modeling of critical components to identify potentially unsafe feature interactions [5]. This approach provides some advantages over the use of feature diagrams. Feature diagrams can document identified interactions but fail to indicate the feature interactions that are safety-critical. The product-line SFTA, however, aids us in identifying those feature interactions that can cause

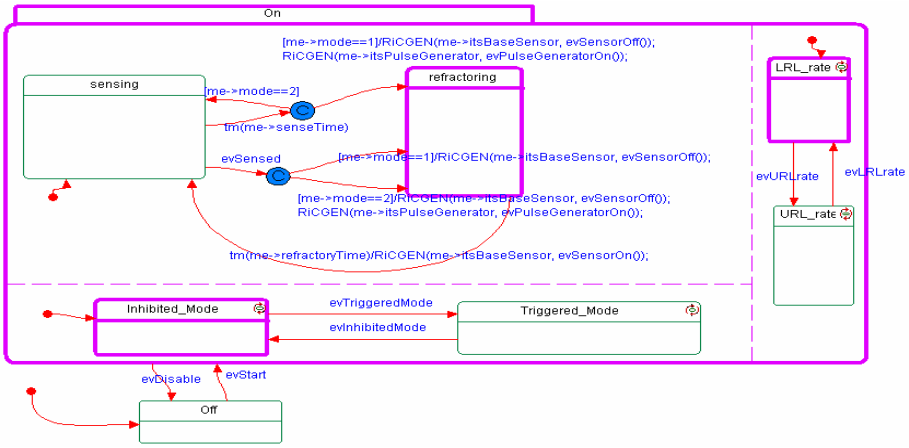


Figure 5. Animated PacemakerController Statechart in the Evolved Product-Line State Model

top-level failure events (i.e., those feature interactions that are safety-critical). We found that using the product-line SFTA greatly reduced the number of feature interactions that had to be investigated. Focusing on critical interactions makes our approach more amenable to use in an industrial setting.

The use of a state-based modeling approach for safety analysis during evolution is advantageous because it can both build on and extend the product-line fault tree analysis. Unlike SFTA, an executable state-based model can analyze and model the timing/ordering of failure events to determine their possible safety implications. In addition, we found that because the SFTA is a static asset, it lacks the ability to animate and explicitly show how a safety property may be violated. The use of an executable state-based model, however, allows the simulation of the behaviors described by the requirements in the fault tree to illustrate the violation of a safety property.

We found that although Rhapsody's executable state-based models support real-time notions, as required in our pacemaker case study, it cannot enforce exact real-time measurement. Thus, the state-based modeling technique described here is not suitable for testing border time values; rather, it can handle testing the ordering logic and relative timing of failure events.

The exploration of alternative new software behaviors in the state-based model to prevent or mitigate the violation of a safety property allows immediate feedback on whether proposed, new safety requirements will indeed guard against the violation of the safety property. Moreover, the executable state-based model, unlike a SFTA, can explore multiple solutions to come up with a reliable and easy-to-implement mitigation strategy. This then drives the updating of the product line's requirements to include the new safety requirements. Such feedback is impossible to ensure using the SFTA alone. Thus, the inclusion of a state-based modeling safety analysis approach, as described here, may improve the safety case that a safety-critical product line must make when requiring certification from an outside governing body.

Reuse of the product-line SFTA as well as of the product-line state-based models constructed during the initial development of the product line, occurred during system evolution in this work. Although, some updates were required to accommodate the new features introduced, large parts of the previously developed safety analysis assets could be reused.

The use of DECIMAL [25], the product-line requirements documentation and analysis tool, coupled with the use of PLFaultCAT [10], the product-line SFTA tool, improved the traceability of the requirements to the components of the state-based models. The leaf nodes of the fault trees constructed in PLFaultCAT are associated with the commonalities and variabilities of the product line, and the state-based models are derived using information from the SFTA. Additionally, the use of PLFaultCAT to identify those feature interactions that may be safety-critical, and therefore should be analyzed using state-based models, helps maintain the traceability of requirements to the state-based safety analysis as the product line evolves.

6. Conclusion

Product-line engineering presents an advantageous approach to developing software systems because the reuse can reduce the development time and cost. Yet, handling product-line evolution is more complex than in traditional software systems because changes to the software requirements may affect or even compromise the various safety properties of multiple products. In particular, the analysis of feature interactions is important because, during evolution, the new features introduced into a product line may have unknown and unsafe interactions with the existing features.

This paper illustrated an approach, built on our previous work with stable product lines, to performing a safety analysis on an evolving product line using a product-line software fault tree analysis to direct state-based modeling. The paper detailed and demonstrated a tool-supported technique to: 1. link product-line requirements to their state-based model components; 2. identify and analyze safety-critical feature interactions; and 3. modify and reuse product-line state-based models to analyze the new features added as a result of evolution. This technique utilized a product-line software fault tree analysis to avoid and manage the complexity of feature interactions.

Future work includes refining our technique and developing further tool-support to provide further guidance in the application of this technique.

7. Acknowledgements

This research was supported by the National Science Foundation under grants 0204139, 0205588 and 0541163. We also thank Telelogic for the use of I-Logix's Rhapsody and TestConductor tools.

8. References

[1] Bennett, K. and Rajlich, V., "Software Maintentance and Evolution: A Roadmap", *The Future of Software Eng.*, ACM Press, pp. 75-87, 2000.

[2] Booch, G., et. al.., *The Unified Modeling Language User Guide*, Addison-Wesley, 2005.

[3] Campbell L, et. al., "Automatically Detecting and Visualizing Errors in UML Diagrams", *Requirements Eng. Journal*, Springer-Verlag, pp. 264-287, 2002.

[4] Clarke, E. M. and Wing, J. M., "Formal Methods: State of the Art and Future Directions", *ACM Computing Surveys*, 28(4):626-643, 1996.

[5] Clauss M., "Modeling variability with UML", *Proc. Net.ObjectDays 2001 Workshop on Generative and Component-Based Software Eng.*, pp. 226–230, 2001.

[6] Clements, P. and Northrup, L., *Software Product Lines: Practices and Patterns*, Addison-Wesley, 2001.

[7] Czerny B. J., and Heimdahl, M., "Automated Integrative Analysis of State-based Requirements", *Proc. 13th Int'l Conf. Automated Software Eng*, pp. 125-134, 1998.

[8] de Lemos, R., "Safety Analysis of an Evolving Software Architecture", *Proc. 5th Int'l Symp. on High Assurance Systems Eng.*, pp. 159-167, 2000.

[9] Dehlinger, J. and Lutz, R. R., "A Product-Line Approach to Promote Asset Reuse in Multi-Agent Systems," *Software for Multi-Agent Systems IV*, Lecture Notes in Computer Science 3914, pp. 161-178, 2006.

[10] Dehlinger, J. and Lutz, R. R., "PLFaultCAT: A Product-Line Software Fault Tree Analysis Tool," *Automated Software Eng. Journal*, 13(1):169-193, 2006.

[11] Deng, G., et. al., "Addressing Domain Evolution in Model-Driven Software Product-Line Architectures", *Proc. Workshop Model-Driven Development for Software Product Lines: Fact or Fiction?*, 2005.

[12] Doerr J., Requirements Engineering for Product Lines: Guidelines for Inspecting Domain Model Relationships, Diploma Thesis, University of Kaiserslautern, 2002.

[13] Ellenbogen, K. A. and Wood, M. A., *Cardiac Pacing and ICDs*, Blackwell Publishing, 2005.

[14] Feng, Q. and Lutz, R. R., "Bi-Directional Safety Analysis of Product Lines," *Journal of Systems and Software*, 78(2):111-127, 2005.

[15] Gomaa, H., *Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures*, Addison-Wesley, 2005

[16] Harel, D. and R. Marelly, *Come, Let's Play: Scenario-Based Programming Using LSCs and the Play-Engine*, Springer, 2003.

[17] Kang, K., "Software Product Line Research Topics," *Proc. 10th Int'l Software Product Line Conf.*, 2006.

[18] Leveson, N. G., *Safeware: System Safety and Computers*, Addison-Wesley, 1995.

[19] Lions, J. L., "Ariane 5: Flight 501 Failure Report", http://sunnyday.mit.edu/accidents/Ariane5accidentreport.html, (Current January 2007).

[20] Liu, J., Dehlinger, J. and Lutz, R. R., "Safety Analysis of Software Product Lines Using State-Based Modeling," To appear *Journal of Systems and Software*.

[21] Liu, J., Dehlinger, J. and Lutz, R. R., "Safety Analysis of Software Product Lines Using State-Based Modeling," *Proc. 16th IEEE Int'l Symp. on Software Reliability Eng.*, pp. 21-30, 2005.

[22] Lu, D and Lutz, R. R., "Fault Contribution Trees for Product Families", *Proc. 13th Int'l Symp. Software Reliability Eng.*, pp. 231-242, 2002.

[23] Lutz, R. R., "Software Engineering for Safety: A Roadmap," *Proc. Conf. on the Future of Software Eng.*, pp. 213-226, 2000.

[24] "Model Driven Development for Real-Time Embedded Applications", *Rhapsody Family Brochure*, http://www.ilogix.com/rhapsody/rhapsody.cfm, (Current January 2007).

[25] Padmanabhan, P. and Lutz, R. R., "Tool-Supported Verification of Product-Line Requirements", *Automated Software Eng. Journal*, 12(4):447-485, 2005.

[26] Prehofer, C., "Plug-and-Play Composition of Features and Feature Interactions with Statechart Diagrams", *Proc. 7th Int'l Workshop Feature Interactions in Telecommunications and Software Systems*, 2003.

[27] Schmid, K. and Verlage, M., "The Economic Impact of Product Line Adoption and Evolution," *IEEE Software*, 19(4):50-57, 2002.

[28] Schwanke R. and Lutz, R., "Experience with the Architectural Design of a Modest Product Family", *Software Practice and Experience*, 34(13):1273-1296, 2004.

[29] Svahnberg, M. and Bosch, J., "Characterizing Evolution in Product Line Architectures", *Proc. of the 3rd annual IASTED Int'l Conf. on Software Eng. and Applications*, pp. 92-97, 1999.

[30] Weiss, D. M. and Lai, C. T. R., *Software Product-Line Eng.*, Addison-Wesley, 1999.

Motion Planning System for Minimally Invasive Surgery

Hanees Haniffa, Jerzy Rozenblit, PhD, Jianfeng Peng
jr@ece.arizona.edu
The Department of Electrical and Computer Engineering
The University of Arizona

Allan Hamilton, MD, FACS, Mohamad Salkini, MD
allan@surgery.arizona.edu
Arizona Surgical Technology and Education Center
The University of Arizona

Abstract

Minimally invasive procedures are highly effective when performed by well trained surgeons. However, with the subjective nature of surgical training and performance assessment, it is difficult to determine when a trainee surgeon has attained a satisfactory level of competency. We propose a computer-based training and performance assessment system where we apply configuration space based techniques to determine optimal paths for the maneuver of surgical instruments to perform predefined tasks.

1. Introduction

Medical training currently lags behind in the application of state-of-the-art technologies to enhance safety and efficiency. The emergence of new surgical techniques enhanced by sophisticated surgical tools and imaging systems, calls for a shift in paradigm in medical practice, to incorporate more computer-based training, assessment, and guidance and warning systems.

We focus our work on training and performance assessment in laparoscopic surgery. In laparoscopy surgeons lose many of the tactile and visual cues that they rely upon in conventional surgery. We propose a design that addresses many of the limitations of existing solutions and advances the state of the art in surgical training and assessment. We propose a computer-based platform where sensor tracking permits motion information to be gathered about surgical instruments used in a procedure. Our design features the embedding of micro-sensors into the instruments employed for training. The detection and recording of instrument movement would permit our system to measure a trainee's progress in acquiring psychomotor skills and compare these data to normative databases.

2. Training System Prototype

The system prototype consists of an enclosure with openings for instrument insertion. Tasks are performed within this enclosure using a video feed. Motion sensors mounted on the tip of each instrument gather data in real-time. We use the microBIRD 3D sensing system developed by Ascension Technology Corporation for this purpose (Ascension Technology Co. 2006). The sensing system includes a magnetic field transmitter, two position sensors (1.3mm in diameter) and a PCI interface data processing card. The measurement rate is 68.3 Hz with linear accuracy of ±1.4mm and rotational accuracy of ±0.5 degrees. The transmitter remains fixed to provide a Cartesian frame of reference for position tracking.

We have designed exercises that involve the manipulation of objects within an enclosed space. These are designed to enhance psychomotor skills, depth perception and 3D visualization, camera handling, familiarity with instrument ergonomics and dexterity. A simplified example of such an exercise may be the placement of a rubber band across two hooks as shown in the cross-sectional diagram below. More sophisticated exercises involving the use of synthetic tissues and organ models to perform basic surgical tasks such as suturing, dissection, translocation, etc. have also being developed.

Figure 1: Basic skills exercise

3. Performance Assessment

We employ motion planning methods to determine the optimal path of the instrument from the current position to the desired goal state. Specifically, we use potential field methods to model the work space as a region influenced by an imaginary force field. The field would be attractive (or have lower potential) at the goal states and be repulsive (or have higher potential) at obstacles and pre-defined "no-fly" zones, as shown in figure 2. The tips of the instruments, in our case would be analogous to robots, and would be represented by points in the configuration space (C-space), which is the region within which the robot navigates. We imagine an artificial potential field U, influencing the motion of the instruments around the goals positions in C-space. We compute the artificial force (attractive towards the goal state and repulsive towards forbidden regions of C) given by

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

$$\vec{F}(q) = -\nabla U(q) \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots(1)$$

where

$$U(q) = U_{attractive}(q) + U_{repulsive}(q) \quad \ldots\ldots\ldots(2)$$

and

$$\nabla U(q) = \begin{bmatrix} \frac{\partial U}{\partial x} \\ \frac{\partial U}{\partial y} \end{bmatrix} \quad \ldots\ldots\ldots\ldots\ldots\ldots\ldots(3)$$

Performance assessment will be based against the "steepest descent" path. Movement towards the steepest descent path will earn credit and movement towards obstacles will impose penalties. An overall score will be computed upon completion of the procedure. The score will also be affected by the time taken to complete the procedure successfully and the number of collisions with obstacles or other instruments and for intrusions outside the C-space region.

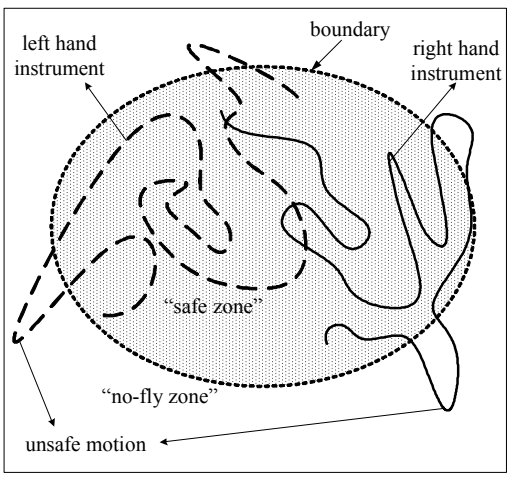


Figure 2: Motion tracks

4. Experiment

We discuss the potential field for the example in figure 1 above. Each instrument is modeled as a separate robot maneuvering in the given C-Space and a separate potential field defined for each instrument. We will focus on the potential field for the right instrument. The left hook will be an obstacle for the right instrument, with the right hook being the goal state in this exercise. The potential field is as represented in figure 3 below.

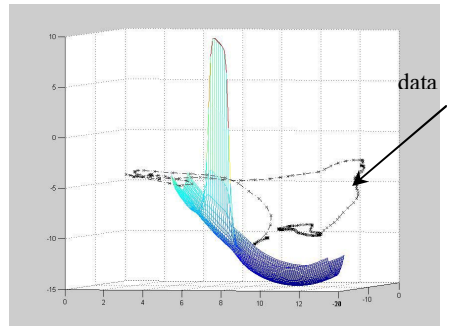


Figure 3: Potential field

5. Results

The collected data was plotted and is shown by the series of points in figure 3. The visual representation of the results of this preliminary experiment demonstrates the principle behind our approach. The collected data is compared against the force exerted by the potential field with respect to current and next positions. For any given current position, an ideal next position is dictated by the potential field. When a trainee moves in the direction of the potential field positive points are awarded and points taken away for movement in a different direction. This paper presents a work in progress and the results are preliminary proofs of concept. The poster version of this paper will contain the results and conclusions drawn from currently ongoing trials.

6. Conclusion and Future Work

The above experiment demonstrates the principles behind our approach. Future work will focus on expanding on the initial idea by refining the development of potential fields, using organ and tissue models for more complex exercises involving suturing, and dissection and exploring the feasibility of incorporating MRI images to define the configuration space.

References

[1] Shortliffe, E. H. 1976. "Computer-Based Medical Consultations: MYCIN" American Elsevier Publishing Co., Inc.,

[2] Ledley, R. S. 1969. "Practical Problems in the Use of Computers in Medical Diagnosis" Proceedings of the IEEE, Vol. 57, No. 11, 1900-1918

[3] Finlay, P.A. and Ornstein M.H.1995 "Controlling the Movement of a Surgical Laparoscope" IEEE Engineering in Medicine and Biology, 289-291

A Process Module to Pre-Process Requirements for Architecting

Matthias Galster
Department of Electrical and Computer Engineering
University of Calgary
Calgary, AB, Canada
mgalster@ucalgary.ca

Armin Eberlein
Department of Computer Engineering
American University of Sharjah
Sharjah, United Arab Emirates
eberlein@ucalgary.ca

Mahmood Moussavi
Department of Electrical and Computer Engineering
University of Calgary
Calgary, AB, Canada
moussam@ucalgary.ca

Abstract

Software architectures have a significant impact on software quality. However, building architectures is a non-trivial task. In this paper, we present a process module for pre-processing requirements before developing the architecture. The benefit of this module is that it "prepares" requirements for architecting: It helps elicit additional technical and process-related information based on requirements. This information will be useful during the creation of architectures. The module can be used in the context of different architecture processes.

1. Introduction

Computer-based systems must implement functional and non-functional stakeholder requirements. Software architectures develop these requirements into an overall structure of the future system. In this paper, we present a process module that helps develop architectures by pre-processing requirements for architecting.

Our work is motivated by existing problems in requirements-based architecting, in particular the missing *relationship* between requirements and architectures [1]. Our objective is to provide means for

- eliciting architecture-relevant information from requirements,
- evaluation of the importance of a requirement from an architectural perspective,
- grouping / clustering of requirements to determine common architectural properties in requirements and strategies for dealing with these requirements.

In the context of software architecting *Global Analysis* [2] was proposed to analyze the global impact of technological, product, and organizational factors on software. In contrast, our approach is applied on a lower level of single requirements and uses a more fine-grained attribute structure.

2. The approach

The proposed approach takes atomic requirements (ATRs [4]) from the requirements specification as input (see Figure 1). This ensures that all requirements are at the same level of abstraction. Moreover, it also allows to cover non-functional software properties: Non-functional aspects are often decomposed and formulated as functional properties before they are satisfied during development [3].

The output of the process module is a set of requirements groupings and clusters. The grouping helps develop technical strategies for requirements (i.e., *what* to do with requirements) when developing architectures. The clusters provide information on process strategies (*how* to do it).

2.1. Process steps

In the *first* step we identify requirements in the initial requirements set for which we already have architectural knowledge in the developing organization. Such requirements are not considered in the following steps 2 – 4. This allows us to do pre-processing only for a subset of the original requirements. For requirements which are not pre-processed, a separate solution space is created, using the existing architectural knowledge. This solution space is later integrated with the solution space generated for pre-processed requirements.

The *second* step is a more fine-grained analysis of requirements for which no architectural knowledge exists. It is based on a set of values for requirements attributes assigned by the requirements engineer for each requirement. The attributes are described by a meta-scheme: textual description of the attribute, its

Proceedings of the 14th Annual IEEE International Conference and Workshops on the Engineering of Computer-Based Systems (ECBS'07)
0-7695-2772-8/07 $20.00 © 2007 IEEE

weight, data type and value range, the means of assessment, the relation of the attribute to the requirements engineering (RE) process, the reason why the attribute is important, and the type of information provided by an attribute (either *technical* or *process* information). So far, we have identified 10 standard requirements attributes: effort, risk, complexity, premature design, interdependencies, volatility, implied hardware constraints, addressed aspects (e.g., user interface, data), priority, and degree of innovation.

The *third* step is the grouping of requirements, based on all attributes. Each requirement is assigned to one value group of an attribute (e.g., attribute "volatility" has possible values high, medium, low).

The *fourth* step is the classification of requirements using clustering techniques. Classification is only done based on attributes which provide process information.

Once groups and clusters are created, the developer can define specific strategies for groups and clusters. Moreover, groups and clusters help in directly deriving architectural artifacts.

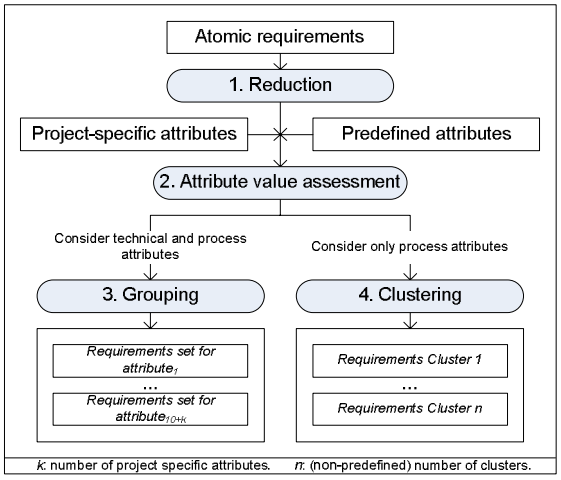


Figure 1. The process module

2.2. Properties of the approach

Adaptability. Standard attributes for pre-processing are provided. Project-specific attributes can be defined using guidelines following the aforementioned meta-scheme.

Iterative pre-processing. As requirements change, redoing the pre-processing is necessary. This is done based on three factors to avoid pre-processing of all requirements: 1) state of requirement (e.g., approved, implemented), 2) impact of change on other requirements as identified through requirements traceability, 3) cluster membership of requirement.

Interfaces. The process module has 3 entry- and exit points: 1. ATRs as *input* are taken as one result from RE. 2. *Step 2* is performed based on any form of knowledge in the organization which can range from formal to informal knowledge. If no knowledge exists no reduction is possible. 3. The *output* (groupings / clusters) provides information that is used for addressing requirements in their architectural context.

General usability. Our approach can be used within generic process models and standards which deal with requirements and architectural design, e.g., the waterfall model, or system life cycle standards (e.g., ISO/IEC 15288).

3. Conclusions and future work

Pre-processing requirements prepares requirements for architecting. The approach has been applied to an example from the real-time domain (cruise-control system). From this, we can draw several conclusions:

- Assessing values of some attributes might be related to *Requirements Analysis*.
- Performing the approach 1) leads to a deeper analysis and understanding of requirements from an architecture-centric perspective, 2) provides information to be considered in different architectural views, and 3) unveils architectural structures.

Our future work includes

- doing extensive examples (also using tool support),
- creating toolkits for each requirements situation (expressed in groupings / clusters), and
- integrating the process module in a more comprehensive process framework to generate architectures based on requirements.

4. References

[1] L. Bastos, J. Castro, and J. Mylopoulos, "Deriving Architectures from Requirements," in Proceedings of the *14th IEEE International Requirements Engineering Conference*, Minneapolis, MA, 2006, pp. 332-333.

[2] C. Hofmeister, R. Nord, and D. Soni, *Applied Software Architecture*. Reading, MA: Addison-Wesley, 2000.

[3] V. Lakshminarayanan, W. Liu, C. L. Chen, S. Easterbrook, and D. E. Perry, "Software Architects in Practice: Handling Requirements," in Proceedings of the *CASCON 2006*, Toronto, Canada, 2006, pp. 329-332.

[4] H. Salzer, "ATRs (Atomic Requirements) Used Throughout Development Lifecycle," in Proceedings of the *12th International Software Quality Week (QW'99)*, San Jose, CA, 1999.

Author Index