Modular, Hierarchical Modeling Concepts for Support of Heterogeneous Systems Design

Christoph Schaffer

Institute of Systems Science
Johannes Kepler University Linz
A-4040 Linz
Austria

Jerzy W. Rozenblit

Dept. of Electrical and Computer Engineering
The University of Arizona
Tucson, Arizona 85721
U.S.A

Abstract

Design of computer based systems (CBS) requires support tools and techniques that can adequately manage the complexity of the underlying engineering activities and processes. In this paper, the primary focus is on the design aspects of the engineering of computer-based systems (ECBS) process. More specifically, it is demonstrated how systems theory and structured knowledge representation concepts can be used to make the system level design more manageable and efficient.

Keywords: heterogeneous systems design, knowledge based design, hardware/software partitioning, structured knowledge representation.

1 Introduction

Over the years, various engineering domains such as electronics, mechanics, aerodynamics or computer science have developed their own abstraction mechanisms to cope with the increasing systems complexity. This worked well for problems with a clear focus on one specific domain. However, today's systems usually consist of a mixture of electronics, software, and mechanical components. Therefore we have to extend domain specific view to a more general one, i.e., the systems view. Systems level is synonymous with a significant abstraction which allows the designer to carry out multiple, different investigations without being hindered by the constraints imposed by the domain, for instance, by the selected realization technology.

System level design comprises the following main activities:

- specification support
- system level modeling
- system level simulation
- system level analysis
- system level partitioning
- integration into a concurrent engineering environment.

At the systems level, many aspects of the final system will be hidden. Therefore we should adequately address the design complexity and ensure its correctness since faults made at the systems level may have a devastating impact on the subsequent design steps. More specifically, we have to define the functional behavior, the systems architecture, partition the system into hardware, software and mechanical parts, while guaranteeing that all the functional and nonfunctional requirements will be met by the proposed design.

In this paper, we present a multi layered design approach that combines structured knowledge representation methods, knowledge-based techniques, and systems concepts to assist in managing the complexity of the system level design of CBS. The concepts presented here can be used to define a system's life cycle that addresses the following needs:

- abstraction to reduce complexity,
- early feedback through simulation,
- rapid prototyping of CBS,
- support design with a knowledge representation scheme, (therefore multiple steps in the life cycle can be nearly automated),
- design of highly reusable components, and

0-7803-2531-1/95/$4.00

- support of hardware/software (HW/SW) partitioning

Our approach complements recent developments that emphasize the use of systems concepts in the engineering of computer based systems [8, 9].

2 General Considerations on Systems Level Design

The increasing complexity of computer based systems and the decreasing time-to-market factor make it necessary to improve and, perhaps, partially automate some of the activities in the systems engineering process.

Highest gains can be achieved if it is possible to reuse previously developed components in new designs. In the hardware domain, reusability is supported by many off-the-shelf components. In software, the object-oriented paradigm contributes to higher reusability. In the 1970s, IBM had identified (through their System Network Architecture (SNA)) [7] the need for design abstraction by introducing several network layers. Thus is it was possible to define a standard which guaranteed that network functions could be reused by different systems.

While it is clear that the need for reusability is well recognized in various domains, it is also true that large numbers of reusable components cannot be handled efficiently without the support of a knowledge base. This is due to the multiplicity of component variants, various decompositions, and potentially combinatorial number of possible ways in which they can be coupled. One can envision design as a search process that operates on the vast space of components and their interdependencies, and generates a system by selecting, coupling, and relating various modules so that the overall design objective is met. Therefore, effective procedures are needed that can manage the complexity of both the underlying design spaces and the search procedures.

When we examine conventional system level design processes, we observe that the reusability of components is not a high priority issue. Typically, the systems are built from scratch, even if the new system is very similar to an already existing one. This is due to system designs being very often strongly coupled to the underlying architecture. Therefore, if a system were designed to be executed as software, for instance, on a 68000 processor, it might be difficult or even impossible to reuse the design and run it on an ASIC as pure hardware. Both systems behave in the same way; however their speeds may be different for different realizations.

By introducing a layered partitioning (LP) model, which is based on the work done in [5, 6], we show how to facilitate design of highly reusable components.

3 System Architecture Specification

We employ the notion of an architecture as a kernel structure for building several partitions, each with the goal of mapping the required functionality to a physical entity. Based on the approach presented in [2], we will discuss the system, hardware, and software architecture.

Systems architecture: At this level we consider the architectural properties which are independent from the realization technology used. Issues such as the geographic distance between components, the topology (e.g., bus, star, ring etc.), whether a component is mobile or stationary, are considered at this layer.

Software architecture: The software structure is addressed here, e.g., the operating system used to run the software on, modularization, etc.

Hardware architecture: Within the hardware architecture we define the needed hardware components (microprocessors, ASIC, off-the-shelf components, etc.) which are needed to fulfill all the functional and nonfunctional requirements.

To make a clear separation between the components at the different architectural layers we employ the terms system component, software component, and hardware component. We strive to clearly separate the components and introduce a hierarchical reference architecture. This is not a trivial task as there is a wide latitude in how the real problem can be addressed. For example, consider the small system depicted in Figure 1.

The three interacting components can be realized in a variety of ways. If we decide to realize all the components in hardware, we have to realize the communication channel between the components in hardware, too. However, if we decide to realize them all in software the channel can be realized by a queuing mechanism. The most complicated case will occur when we want to realize two components in software and only one in hardware. Then, a software

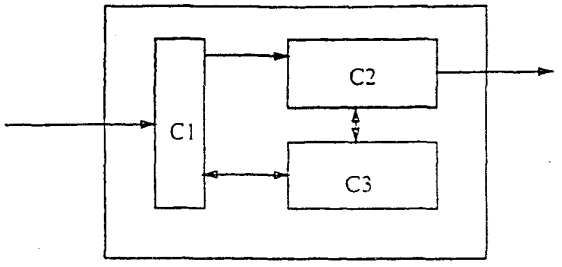

Figure 1: System of interacting components

and a hardware channel are required. We will also have to convert software signals to a representation that can be understood by the hardware component and vice versa.

Therefore, each time we change the system partitioning, it is likely that we will need another realization of a communication channel, or that we will have to introduce new functionality by the required conversions. To overcome these problems we will use the layered partitioning model introduced in the ensuing section.

4 Layered Partitioning (LP) Model

As shown in Figure 2, the LP model consists of the component, port, and channel layers. At each of these layers we can find technology free components which correspond to the system components defined above.

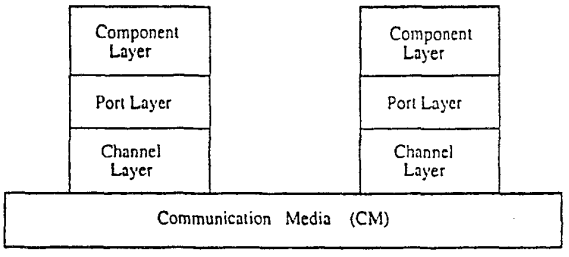

Figure 2: LP model

Component Layer: When we refer to the system architecture before any partitioning into hardware, software, or mechanical parts, then all the components available in the system architecture will also be available within the component layer. To distinguish these components from the components of the channel and port layers we call them *component layer components* (COLC).

Channel Layer: The channel layer is responsible for transport of data from among COLCs. The components used here are *called channel layer components* (CHLC). The CHLC uses a communication medium (CM) to transport data. CHLC is introduced after we do the first hardware/software partitioning.

Port Layer: At this layer we define all the functionality needed for data type conversion. These kinds of components are called *port layer components* (PLC).

We emphasize that although the CHLC and PLC will be determined by partitioning the system into hardware and software, they still belong to the realization technology free system architecture. These components are only a means to realize a proposed partitioning — we still must decide if a PLC or a CHLC should be realized in hardware or in software.

5 The System Entity Structure (SES)

To adequately represent the multitude of a computer-based system's elements we propose to use a knowledge representation scheme called system entity structure (SES) and its enhanced, frame-based representation called Frases [3, 4]. Through SES, we can express an object's decomposition hierarchy, the taxonomy of design elements, the constraints on coupling of objects identified in decompositions, and the constraints on selection of elements given by the taxonomic relationships.

A system entity structure is a labeled tree. Nodes of the tree are called *entities, aspects, specializations, and multiple decompositions.* An entity node represents a real world object (which can be independent or can be identified as a component of some decomposition or specialization of a real world object. Aspects are decomposition types, or in other words decomposition views. Specializations is a taxonomic relationship, i.e., a particular way of classifying an entity. A multiple decomposition is a special type of decomposition (aspect) used to represent entities whose number in a system may vary.

As an example (upon which we illustrate the systems engineering concepts discussed in this paper), consider a high level SES representation of a ship

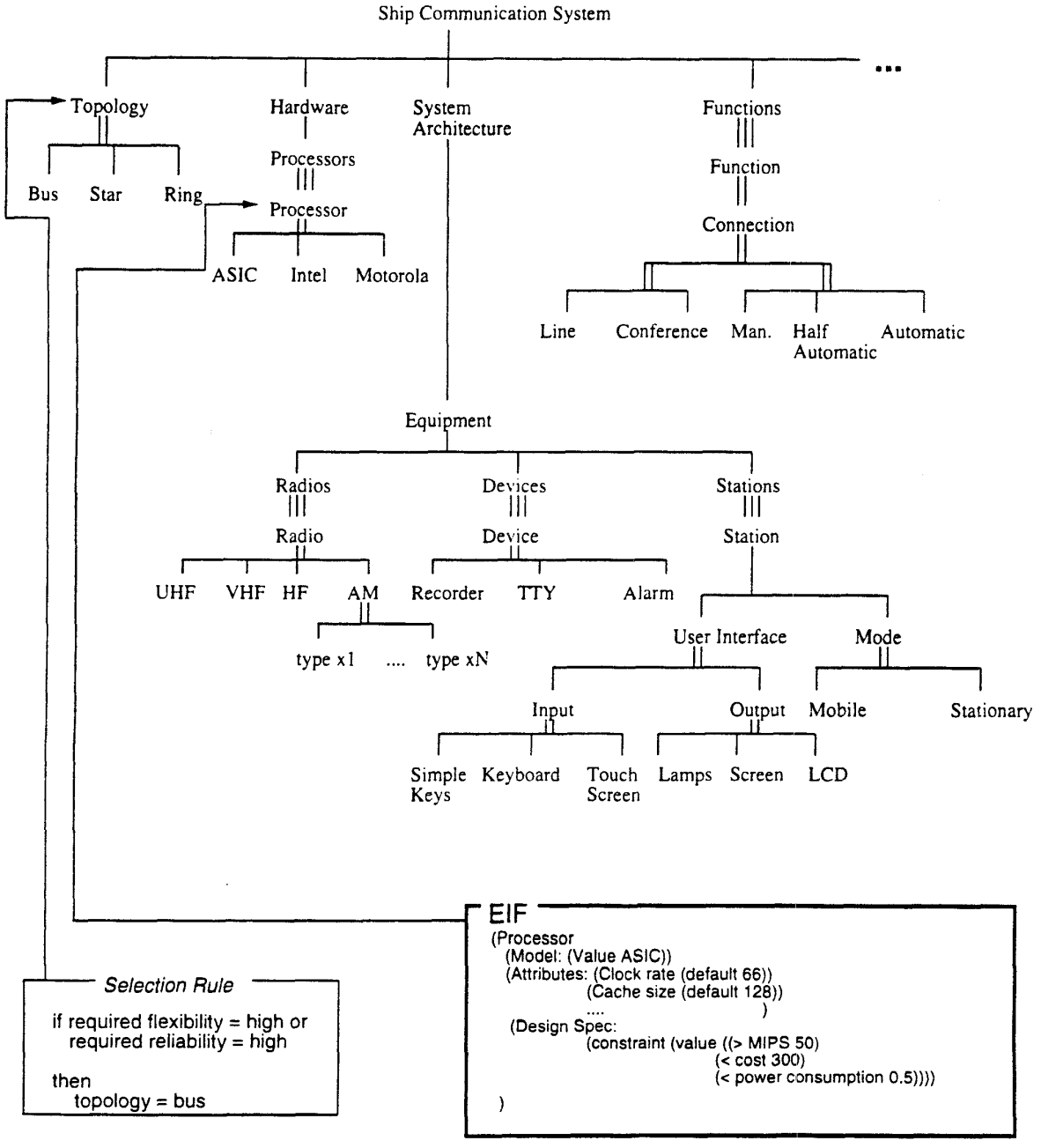

Figure 3: SES representation of a ship communication system

communication system shown in Figure 3. The Ship Communication System entity represents the system to be designed. It consists of major aspects such as Function, System Architecture, Hardware, and a Topology specialization with the entities Bus, Star, and Ring. Systems Architecture is decomposed into Equipment. The Equipment consists of Radios, Devices, and Stations. These are classes of objects. Their instances are Radio, Device, and Station, respectively. (The triple vertical line denotes a special the *multiple decomposition* [4] — a relation that breaks up a class into a set of instances whose number may vary.) Functions are Line and Conference Connections with Manual, Half-automatic, and Automatic types of connection.

An SES specifies a family of possible arrangements of components for the system being designed. Aspects and specializations allow us to specify a family of design alternatives by selecting different components and decompositions. The multiplicity of taxonomic and decomposition relationships in a large design entity structure leads to a combinatorial explosion of possible model alternatives. Therefore, it is necessary to provide procedures that effectively reduce both the complexity of the search process for admissible model structures and the size of the search space itself. We have developed a production rule-based procedure, called *pruning* to address this problem.

Pruning the system entity structure results in a set of alternative design model structure [3, 4]. As described in [4], this is done by defining selection rules for each specialization, and synthesis rules for decomposition nodes of the system entity structure. (An example of a selection rule is shown in the figure.)

To improve the efficacy of design representation, we have augmented SES with a frame-based scheme called Frames and Rules Associated System Entity Structure (Frases). Frases combines the frame, rule-based, and SES knowledge representations. An underlying data structure of Frases is the system entity tree. Each node of the SES tree has a frame attached to it that encompasses declarative and procedural knowledge in a design problem. Such a frame is called *Entity Information Frame* (EIF). An Entity Information Frame (EIF) integrates design knowledge by providing slots for representing design attributes, functional, and procedural knowledge. Thus, the SES layer assures that the decomposition, taxonomic, and coupling relationships are properly specified while each EIF captures the additional design information such as attributes, constraints, requirements, etc. In the figure, we have shown a simple EIF for the entity ASIC Processor. Further details on the Frases representation can be found in [3].

6 The Systems Life Cycle

In the following we proceed to show how the SES and LP models can be used in designing computer based systems. For the sake of brevity, we use a small part of the ship communication system as a illustrative example.

6.1 Specification Support

When starting a new project, a designer typically works with an informal specification of requirements which should be fulfilled by the system. For large scale, complex systems it is difficult to prove that all the requirements are complete and consistent. Life Cycles as presented in [1] handle this problem by building executable models based on the informal requirement specification, and by using analysis (such deadlock, reachability, etc.) and simulation techniques. By using those methods, the designer can determine if there are missing or inconsistent requirements. In addition, by "animating" the models, designers can better comprehend how the system is intended to behave.

In our proposed methodology, the SES is the basis for the selection and configuration of the system's components. In the initial design phases, when boundaries between the system and its environment are defined, the following two cases are considered:

1. The boundary defined by the requirement specification itself. This kind of environment is called the *external environment*.

2. By modeling the external system environment, we better understand the requirements for the system to be built. We recognize that some of them can be realized by components which are available on the market. Therefore, if we use such components, the requirement for the remainder of the system to be built may change, depending on the functionality given by the already selected components. This process can be iterative in that each selection of certain components will result in new constraints that have to propagated onto the remaining selection decisions and so on. The components whose selection is driven by the external environment constitute what we call the *internal environment*.

For illustration, consider the following example. A requirement stipulates that radio communication within the AM frequency band be available. It would be unreasonable to design and build a new AM radio. Instead, we use the knowledge associated with the SES representation of our domain problem to. From the SES, we realize that there are N different types of AM radios available on the market. As mentioned above, each entity has a frame attached to it with attributes such as price, weight, size, power consumption, etc. Based on the attribute values, we select a radio type most suitable for the design problem at hand.

Now, we can now proceed to select an appropriate type of user stations. This requires that a choice of a mode (i.e., stationary or mobile) be made and that this choice be propagated onto the type of input interface. (Choosing the mobile mode of operation restricts the number of keys on the input device to twenty (20) due to both power consumption and space restrictions.) By proceeding in this fashion we can configure the internal environment. Additionally, we can estimate the overall costs, weight, size, and other parameters. This helps in ensuring that some of the nonfunctional requirement can be met.

When we select a specific component, e.g., an AM radio, we also create a COCL that holds a behavioral description of the interface for the selected type of the AM radio.

Having selected one or several possible configurations of the internal environment, we proceed with the functional requirements. Now, the SES representation provides knowledge about the functionality of a system in the domain of interest. In our ship communication system, we can select a type of conference connection form those SES defines as feasible, i.e., manual, half-automatic, and fully-automatic. If we select the fully automatic conference connection, we must choose the specific stations which will be part of the conference cycle. As a result we may arrive at a model shown in Figure 4.

The model has three user stations (one of them is mobile), two radios, and a tape recorder. The system should allow fully automatic 2 point connections between user stations 1 and 2 and a half automatic 2 point connection between user stations 2 and 3. Additionally, it should be possible to record the calls at user station 1. User station 2 should be able to make external communication with an AM and an UHF radio (Remark: not all of this functionality is shown in the SES of the communication system).

Since we assume that the behavior of all the selected components is described by a formal language such as SDL or STATECHART, we will be able to "animate" the system and thus check if the system fulfills all the functional requirements. We can also use deadlock, reachability analysis, and other techniques to determine if the system is functionally correct.

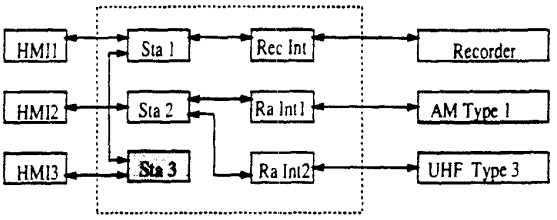

Figure 4: Pruned Substructure

So far, we have only considered a design approach where all of the requested components are already available in the SES representation of the system. However, in many systems we need to build new components which are not available "off the shelf". This situation will be reflected in the specification support phase by building the so called *Specification Models* [5, 6].

Rather than describing the functionality in a very detailed manner, we propose to use an abstract description as a specification for the design of the new component. This abstract description should be available in an executable form [1].

6.2 System Level Modeling

As stipulated above, the *Specification Models* are thought of as surrogates of the unavailable components. In the system level modeling phase, we convert the *Specification Models* to so called *Design Models*. They are used to realize all the requirements defined in the *Specification Model*. Whereas the *Specification Model* specifies what the component has to do, the *Design Models* shows how this behavior is achieved. The requirements in the *Specification Model* will be replaced in the *Design Model* by real parameters determined while designing the component.

Design models allow us to run performance simulation to determine measures such as data rates at the communication channels, to identify time critical parts or bottlenecks within the system.

The next phase in our approach is to determine the geographic distances (geographic partitioning (GP)) between the components (For example, Fig-

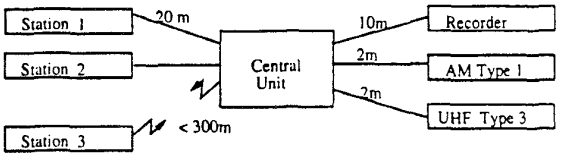

Figure 5: Determining geographic partitions

ure 5 illustrates the distances between the user stations and the radios.) Then, we proceed to select the topology. In the best case, the requirements, geographic partitioning and performance simulation results restrict the search space to one outcome. If there are more possibilities, the best solution is determined by re-evaluating design parameters and running performance simulation again. The best topology can be selected through this iterative process. By validating the design early on, we are able to understand how well the functional and nonfunctional requirements are met by the proposed design. If necessary, the design can be further refined. If problems occur, corrections can be made very early in the life cycle.

The geographic partitioning step introduces various physically distributed partitions. Together with the data rates of the communication channel determined while executing the performance simulation of the system, the partitions are an essential parameter used in the selection of the communication medium (CM). Since the realization of the communication between the components might have a significant impact on the overall system performance and cost, we again attempt to find the best design solution. We search the SES to find possible realization for our communication task.

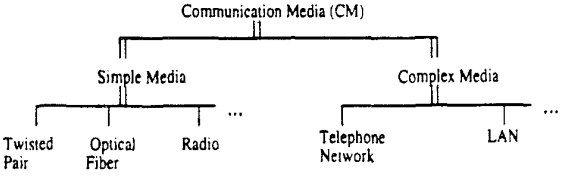

Figure 6: SES of communication media

Figure 6 shows a representation of the communication media (CM). Again the parameters in the entity information frames give us costs, weight, error rate at the medium by a given data rate, signal to noise ratio, etc.

If all the required parameters for selecting the appropriate CM are given by the preceding steps of the systems life cycle, the medium (or media) fulfilling all the requirements can be selected with relative ease. For example, assume a component has to be a mobile station. This will restrict the possible realizations for the CM dramatically.

For CMs we also select associated performance models (Figure 7). They are used for performance evaluation. We can thus determine if the system still fulfills all the performance requirements.

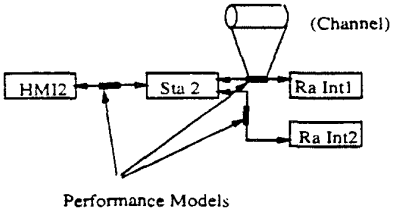

Figure 7: Associating performance models

So far, we have discussed the introduction of physical partitions by executing only the GP step. In addition to GP, we must also consider the hardware/software partitioning.

6.2.1 Hardware Architecture

We use the data gathered while running performance simulation to determine some of the requirements for the needed hardware. We may identify a time critical component, e.g., one that has to deliver a signal every microsecond — a potential candidate to be realized in hardware (HW). The partitions generated while executing the GP step can now be used as parameters for selecting the needed hardware components. Having an isolated view of such partitions, helps us determine the appropriate input and output data rates.

Here, an SES of the hardware can be used to decide on appropriate elements. This SES will also hold different performance models for all of the HW components. When realization components are decided upon, we can begin to check if the all the functional as well as nonfunctional requirements can be met.

Figure 8 shows our system with all the functionality (except for the mobile component) mapped to one single processor. The mobile component is

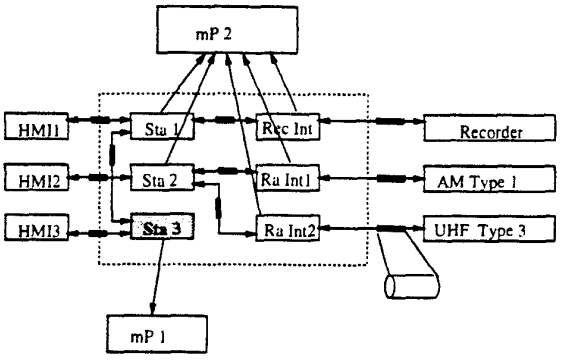

Figure 8: Introducing a hardware architecture

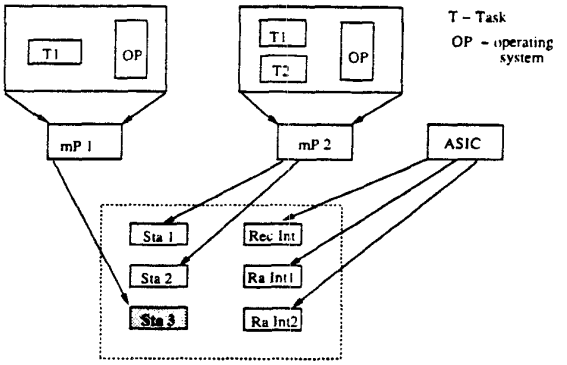

Figure 9: Hardware and software architecture

mapped onto its own microprocessor. This mapping can be achieved easily by using a tool environment such as the one described in [1]. By running performance simulation, we now examine the effects of the hardware architecture on the overall system performance. A variety of hardware realizations can be tested in this manner. For example, an alternative scenario would be to realize the radio and recorder interfaces by an ASIC. In that case we have to use the performance model of an ASIC as shown in Figure 9.

6.2.2 Software Architecture

To run our system on a microprocessor, we need an operating system (OS). The requirements dictate the choice of the best OS. Typical questions that we must answer are: "Do we need a real time operating system, how many processes, tasks have to be executed and how will the system behave if the number of tasks and therefore the overhead due the operating system increases?"

We propose to use an OS representation SES with various operating systems, performance, costs and other measures in this phase. Coupling rules will tell us if a selected operating system can be used in conjunction with a specific microprocessor. Again we will use performance models, and by mapping COLCs to the different software components (tasks, processes, etc.) we will be able to determine how the operating system behaves on a specific microprocessor.

Figure 9 shows the chosen software architecture for our ship communication system. Stations 1 and 2 are executed as individual tasks on microprocessor 2 whereas the mobile station is executed on microprocessor 1.

6.2.3 Identification of Channel Layer Components

After we have introduced the first hardware and/or software architecture and determined that all the requirements can be met by this architecture, we proceed to identify the channel layer components (CHLC). In the ship communication system, two microprocessors are used to realize the required functionality. Let us further assume that the software architecture is already specified.

Using the SES representation, we determine the required communication media. Since one of the processors holds the mobile station and the distance between both processors might be long (e.g., 200 meters), we must use radio waves as a communication medium. The communication between the software components can be realized by a simple queuing mechanism.

Given the data rate of the channel, the error rate, the required error rate, etc., we can select an appropriate protocol to be used, and subsequently an CHLC which realizes it. Again the SES holds performance models of the CHCL. In the first performance simulation run, we use the models to determine if the CHCL fulfills all the requirements. If this is the case, we supplement COLC with the CHLC. Changes are now required in the hardware and software architecture. It is clear that we need a transceiver to send and receive data to/from the mobile microprocessor. The software architecture need not be changed significantly because we use a standard queuing mechanism.

6.2.4 Identification of Port Layer Components

Having determined all the CHLCs, we address the required port functionality. The ports are used to reconcile the different signal representations. We now need to convert signals sent by the mobile component (software signals) to hardware signals that can be modulated and transmitted. At the receiver, the hardware signals must be converted to software signals. These conversions require functional specifications. We search for them in a domain specific SES.

We use performance models first to determine if all the requirements are met. If they are, we use the selected port layer components (PLC) described by a formal language. We add these PLCs to our system consisting of COLCs and CHLCs. We examine the hardware and software architecture again and we realize the HW/SW signal type conversions (and vice versa) by dedicated IO devices. Simulation determines if all the requirements are met by the proposed design.

If all the requirements are met, we have a fully partitioned system with the requisite microprocessor, memory, I/O devices and communication media. However, it should be emphasized that all our components are still described at systems level.

7 Conclusion

In the design approach presented here, we recognize that the clear separation between system level modeling, simulation, analysis and partitioning may not always be enforced. To allow early feedback, which is essential in a complex design process, we should be able to make many iterations between all these phases. By facilitating such iterations, we can examine the impact of a design decision very early in the cycle.

We plan to refine the design desiderata and phases presented here into a formal methodology for heterogeneous systems design.

References

[1] P. Gerlich, C. Schaffer, V. Debus and Y. Tanurhan. EaSyVaDe: Validation of System Design by Behavioral Simulation, *Proc. of the 3rd ESTEC Workshop on Simulators for European Space Programs*, Nordwijk, November 1994 (in press).

[2] D. Hatley and I. Pirbhai. *Strategies for Real-Time System Specification*, Dorset, New York, 1988.

[3] J.W. Rozenblit and J.F. Hu, Integrated Knowledge Representation and Management in Simulation Based Design Generation, *IMACS Journal of Mathematics and Computers in Simulation*, 34(3- 4), 262-282, 1992.

[4] J.W. Rozenblit and Y.M. Huang, Rule-Based Generation of Model Structures in Multifacetted Modeling and System Design, *ORSA Journal on Computing*, 3(4), 330-344, 1991.

[5] C. Schaffer. An Approach to Design Complex, Heterogeneous Hardware/Software Systems. *Proc. of the International Conference on Computer Aided Systems Technologies*, May 1994, Ottawa Springer Verlag, (in press).

[6] C. Schaffer. HW/SW Codesign - A Systems Level Approach, Internal Report, Johannes Kepler University, Dept. of Systems Theory and Information Engineering, August 1994

[7] M. Schwartz. *Telecommunication Networks: Protocols, Modeling, and Analysis*, Addison-Wesley, 1987.

[8] G. Schweizer and M. Vosss. Managing the ECBS Process - Towards a System Theory for ECBS. *Proceedings of the 1994 Systems Engineering of Computer Based System*, IEEE Computer Society Press, pp. 124-130, Stockholm, May 1994.

[9] B. Thome (Ed.). *Systems Engineering: Principles and Practice of Computer-Based Systems Engineering*, John Wiley and Sons, 1993