Towards an Integrated, Model-Based Codesign Environment

S.J. Cunning, T.C. Ewing, J.T. Olson, J.W. Rozenblit, S. Schulz
Department of Electrical and Computer Engineering,
The University of Arizona
Tucson, Arizona 85721-0104
{scunning|tce|olson|jr|sschulz}@ece.arizona.edu

Abstract

This paper describes desiderata for an environment to implement the model-based codesign methodology. A brief summary of model-based codesign is given, followed by a discussion of related work. The services and capabilities required of a design environment are given within the concept of a CAD framework. The design flow that realizes model-base codesign is presented. Attention is given to the required activities at each step and to the flow of design information between design steps. The design flow described here focuses on real-time embedded systems. It covers engineering activities from requirements documentation to the physical realization of the design. Emphasis is placed on modeling and simulation to support automated design reasoning. Considerations related to tool support and tool integration are also given.

1. Introduction and motivation

Numerous authors point to the deficiencies of the traditional hardware/software design frameworks [1,2,3,4,5,6]. They strongly advocate a process that fosters the integration of hardware and software design perspectives. Therefore, a unified representation is needed for modeling a system independent of its physical realization.

We have proposed a design approach called *model-based codesign* [5,6] that uses stepwise refinement of simulatable models and offers the opportunity to abstract system components at multiple levels of representation. In this methodology, a set of requirements and constraints is obtained for the system to be modeled. The system is then described as an abstract model that is a combination of its structural and associated behavioral specifications. Model components are specified at a high level of abstraction to remain implementation independent.

In model-based codesign, we verify correctness of models through computer simulation. A simulation test setup is called *experimental frame* [7]. It is associated with the system's model during simulation. Such frames specify conditions under which the system's model is observed. Simulation is then executed according to the run conditions prescribed by the frames. At the end of the simulation process a virtual system's prototype is obtained. The design is then partitioned into hardware, software and corresponding interfaces using a process that we call *model mapping.*

Clearly, the efficacy of a design methodology can only be confirmed through its application in a design environment. Although much research has been done on codesign, only two successfully implemented environments have emerged that support heterogeneous embedded systems design.

Recently reported work on the design environment POLIS [1], describes a joint project of the University of California at Berkeley and Cadence Design Systems. Its goal is to enhance the design of control-dominated embedded systems applications. There, the system is first translated into a representation consisting of Codesign Finite State Machines and formally validated. After a manual partitioning by a designer and automated hardware and software synthesis, the generated implementation is evaluated using a co-simulation engine. One benefit of this development environment is the ability to synthesize the formal CSFMs into hardware/software components. The partitioned components can be co-simulated at a cycle level to guide the designer to the next partitioning iteration. The authors restrict themselves to relatively small, real-time control systems applications. Also, POLIS does not provide any hardware/software partitioning algorithm yet.

COSYMA, a design environment developed at the Technical University of Braunschweig [8], focuses on codesign using a dual processor platform for the exploration of the co-synthesis process applications that require intensive data processing. Initially the design is realized entirely in software. If a run-time analysis fails to satisfy the specified timing constraints, processes are then moved to an automatically generated application specific co-processor in an iterative manner. Automatic design support guides the designer in scheduling the software tasks to satisfy hard and soft constraints. Partitioning is supported by simulated annealing - a stochastic optimization algorithm.

At the University of Arizona, we are developing a computer-aided design environment called SONORA. This environment implements the theory-based tenets of the model-based codesign. We are striving to provide an integrated tool set that will support the design automation of complex, real-time embedded systems. In what follows, we first summarize basic desiderata for an electronic design automation (EDA) framework. Then, we proceed to specify the architecture of the SONORA system and its functionality.

1.1. CAD framework concepts

The term framework in electronic design automation denotes a computer-based, integrated design environment that binds together and supports design tools [9]. In their seminal paper, Harrison et al. [10] define a framework as a set of underlying facilities provided to the CAD tool developer, system integrator, and end user necessary to facilitate their tasks. The CAD Framework Initiative views a framework as a collection of extensible programs/modules used to develop a unified CAD system [9]. In essence, a CAD framework is a software infrastructure that provides a common operating environment for computer aided design tools. A framework should enable users to launch and manage tools; create, organize, and manage data; graphically view the entire design process; and perform design management tasks such as configuration and version management. Among the key elements of a CAD framework are platform independent graphics and user interfaces, inter-tool communications, design data and process management, and database services. For a summary of the evolution of the framework concept we refer the readers to [10,11].

Here we briefly enumerate the postulates for framework functionality that provide a comprehensive abstraction of the universally accepted layers of design support. Bhat and Taku [9] proposed this model to address the following issues of data integrity, distributed design data management, design process management, and configuration management. Bhat and Taku stipulate that seven functional layers (listed below in the increasing degree of abstraction) be present in a CAD framework:

1. Basic Services: facilitate access to databases in the form of procedural or command interface and interprocess communication.
2. Design Representation: a model for creation, storage, retrieval, and modification of design data.
3. Data Management: facilitates mapping the physical design data onto the logical view of design. This layer also manages the design versions and configurations as well as logical relationships between design objects.
4. Tool Management: models of how to control various design tools. Tracing tool execution is the function of this layer. (See Bretschneider [11] for an extensive treatment of this issue.)
5. Data and Tool Integration: provides facilities for integrating tools independent of the physical, logical, project, object, and tool specific data formats and structures.
6. Design Process Management: offers facilities for modeling the design process. Supports decision-making by aiding the user in design step selection and iteration.
7. Methodology: guidelines, rules, procedures, and methods, which reflect designers' style and philosophies.

Furthermore, a division is drawn between tool frameworks and design process frameworks. Layers 1 through 3 of the hierarchy are usually related to application tools. The remaining layers are primarily concerned with the design process management.

Our goal is to realize the model-based codesign methodology in an integrated design environment that meets the framework requirements stipulated above. The two principal objectives in developing the environment are: a) to provide a integrated tool set for the design of embedded systems, and b) to provide a design flow control and management procedures.

In the following section, we introduce the SONORA environment.

2. The SONORA environment

The SONORA environment is intended to realize the model-based codesign methodology. An abstract representation of this methodology is shown in Figure 1.

The Functional and Behavioral Requirements Specification and Modeling block embodies requirements solicitation and documentation followed by development of an executable model. The Behavioral Simulation and Model Refinement Loop is used to iteratively refine the design model until it is functionally correct. Structural

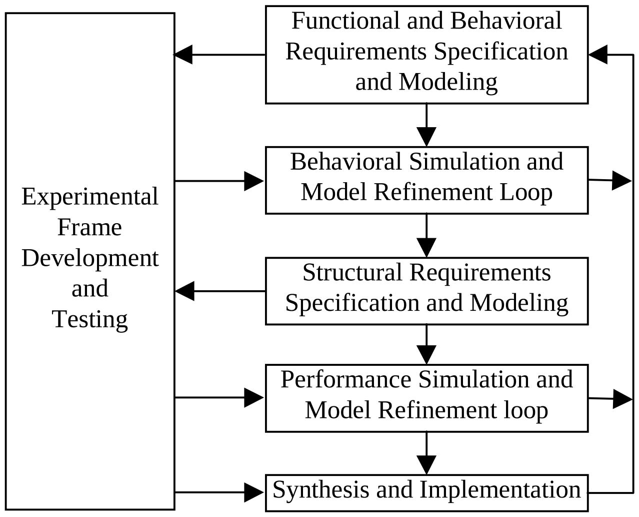
Functional and Behavioral Requirements Specification and Modeling
Behavioral Simulation and Model Refinement Loop
Structural Requirements Specification and Modeling
Performance Simulation and Model Refinement loop
Synthesis and Implementation
Experimental Frame Development and Testing

Figure 1 Abstract design flow

Requirements Specification and Modeling relates physical design constraints to a proposed physical architecture. In the Performance Simulation and Model Refinement Loop the model is enhanced with performance estimates for computation and communication based upon the proposed physical architecture. Synthesis and Implementation involves extracting design information from the models in order to produce a physical prototype. Experimental Frame Development and Testing involves the creation of a set of test cases based upon the system requirements that are used to asses the current design at all stages of the design process.

2.1. Functional and behavioral requirements specification and modeling

Modeling is a natural approach to codesign. There are many tools and environments [12,13,14,15,16,17, 18,19] available on the market that make model creation easier and more understandable using graphical enhancements.

Requirements are solicited from the customer or derived from a perceived market need. The functional and behavioral requirements are documented by using a structured form-based approach that emphasizes the separation between system and interface requirements. These requirements are used as input to the Experimental Frame Development and Testing block. There, they are converted to test cases to be used throughout the design process.

After requirements capture, an initial high level model of the system is created. An initial functional structure for the model is extracted from the system

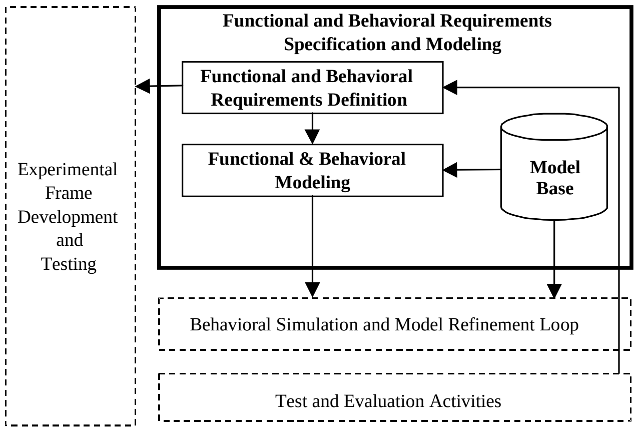
Functional and Behavioral Requirements Specification and Modeling
Functional and Behavioral Requirements Definition
Functional & Behavioral Modeling
Model Base
Experimental Frame Development and Testing
Behavioral Simulation and Model Refinement Loop
Test and Evaluation Activities

Figure 2 Functional and behavioral requirements specification and modeling

requirements. The components of this model are then refined to lower levels of abstraction by the designer with the aid of simulation during the Behavioral Simulation and Model Refinement Loop phase of the design process. A repository of previously created model components is maintained to allow reuse when possible. These model components may be used during initial model creation or later, during model refinement.

The feedback path from the Test and Evaluation Activities demonstrates the iterative nature of model-based codesign. If, at downstream step, an incomplete or inconsistent requirement is detected, the requirements must be refined and effects flowed down.

The behavioral aspects can be represented using Petri [20] nets, StateCharts [14], Discrete Event System Specification (DEVS) [21], and other system specification formalisms.

2.2. Behavioral simulation and model refinement

SONORA allows simulation of the different levels of abstraction contained within the system model. The results of simulation verify behavioral requirements of the embedded system, which can be used in the stages before model-to-realization mapping (i.e., categorizing system model components into types of generic hardware and software). Thus, in parallel to the simulation, we assign classifications to the system model components using a Bayesian Belief Network (BBN) [22,23,24].

The application of BBN to hardware/software partitioning was first introduced by Olson and Rozenblit [25]. The BBN delineates the decomposition of the system model into several levels of abstraction through causal relationships between the component nodes (shown in Figure 4). The ability to iteratively analyze and modify

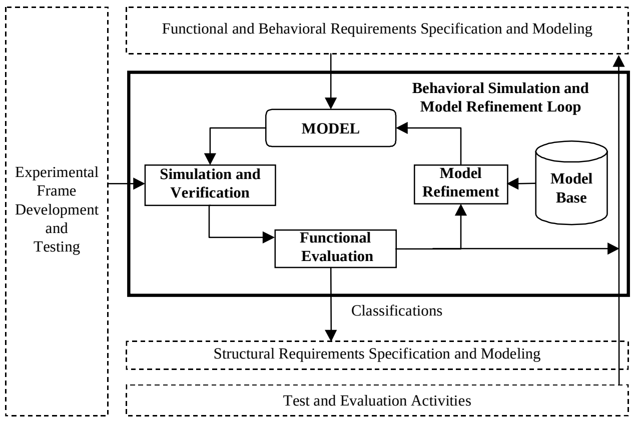
Functional and Behavioral Requirements Specification and Modeling
Behavioral Simulation and Model Refinement Loop
MODEL
Experimental Frame Development and Testing
Simulation and Verification
Model Refinement
Model Base
Functional Evaluation
Classifications
Structural Requirements Specification and Modeling
Test and Evaluation Activities

Figure 3 Behavioral simulation and model refinement

the system model, and the ability to generate classifications of all components are therefore, the objectives of the behavioral simulation loop shown in Figure 3. The requirements necessary for simulation and component classification and a brief description of the simulate-analyze-and-classify loop follow.

The hierarchical nature of the system model requires that experimental frames and interfaces between levels of abstraction be well defined. In addition, the BBN must convert results from simulation runs into evidences that can be introduced to the component nodes. The introduction of evidence shown as E1 and E2 in Figure 4, along with the causal structure of the belief network can be combined to calculate the beliefs of component classifications (e.g., the analysis of a simulation result may introduce evidence in support of implementing a given component in a generic type of hardware or software). To see how these requirements affect SONORA's simulation, let us now examine its functionality.

Before classification can begin, a functional description of the model is created (in a manner similar to the Specification Level Intermediate Format (SLIF) [26]). Next, the BBN is generated with nodes representing functional components, and causal links corresponding to component couplings, function accesses, and functional independence of components. The choice of which values to place inside the conditional matrices associated with each link depends on the communication needs between the given pair of elements, and how tightly their performance is coupled. Once the BBN is created, it can be used to evaluate the current design by incorporating the simulation results as evidences.

During each iteration of the behavioral simulation loop, results are obtained from simulation and converted into evidence that is propagated throughout the BBN.

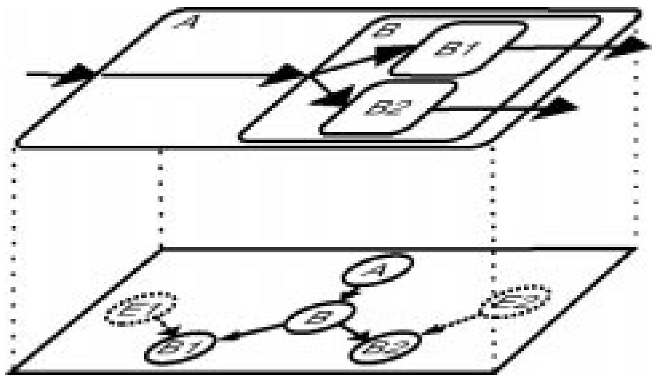

Figure 4 High level model with BBN

The beliefs for each available type of classification are calculated at each component node and the system model (now possibly with some classified components) is altered to reflect the new classifications, as shown in Figure 4 in the model refinement block.

Simulation is performed again, and the process is repeated until the components of the system model reach a level that requires the introduction of structural requirements for any further classification. The result of the refinement of the behavioral simulation is a functionally correct virtual prototype of the design. Each component is assigned to a general classification of a type of hardware or software.

2.3. Structural requirements specification and modeling

Structural requirements, which have been deferred from modeling to this point, are introduced here. These can include non-functional requirements and constraints such as component budget, power consumption, off the shelf components' restrictions, etc. A collection of different design architectures will be used to support an initial structural model and provide a starting point for the

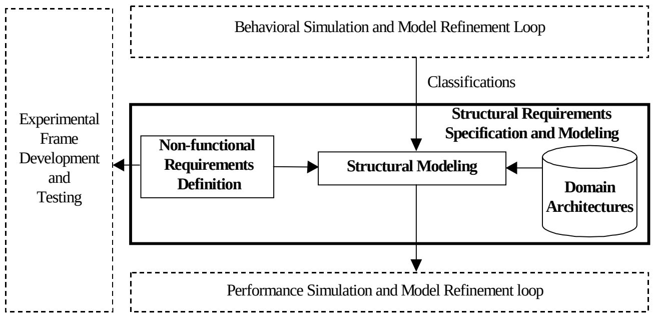
Behavioral Simulation and Model Refinement Loop
Classifications
Experimental Frame Development and Testing
Structural Requirements Specification and Modeling
Non-functional Requirements Definition
Structural Modeling
Domain Architectures
Performance Simulation and Model Refinement loop

Figure 5 Structural Requirements Specification and Modeling

following iterative fine-tuning of a prototype.

The structural aspects of a system can be represented using many different methods. Among these are OMT [27] and System Entity Structure (SES) notation [28,29]. The SES notation allows for the representation of a family of system configurations, which are then pruned to generate specific design instances.

In the structural modeling, we transform our behavioral model by adding the structural aspects of the design which inherently introduces implementation dependent parameters. At the behavioral level, the clearest solution available is a classification of a functional component into a standard type of tangible component, e.g., there exists a 75% belief that a particular function should be implemented in a generic FPGA. The addition of structural information allows the functional component to be mapped into a specific component within a specific solution, e.g., instead of mapping into a generic FPGA, a given component is combined with three other similar classified components to share a single FPGA chip to conserve space. From the virtual model, useful information can be retrieved that has a strong influence on the model mapping, e.g., worst case processing times or suggestions for an efficient instruction set. The structural model should incorporate already existing components and architectures from a design library to foster reuse. In addition, the couplings of the model are propagated in conjunction with the component classifications as interface constraints in the final Synthesis and Implementation block. Timing constraints are propagated to the following structural simulation. This unified model serves as a starting point for the Performance Simulation and Model Refinement block.

2.4. Performance simulation and model refinement

After the system design is completed and evaluated at the virtual model level, a physical realization of the system is carried out by stepwise refinement. The translation from the model to the actual implementation will be automated in SONORA.

For a summary of approaches that automate hardware/software partitioning in codesign we refer the reader to [4]. In our approach, we use performance simulation to finalize and refine the classification choices made during the previous step, i.e., structural modeling. Therefore, rather than assigning components to hardware and software clusters, we group processes according to a broad range of hardware and software classes so that they fulfill the necessary requirements as a component in the final embedded system design. These requirements consist of performance requirements needed to satisfy its hard real-time processing constraints and non-functional requirements, which are evaluated in the performance simulation analysis.

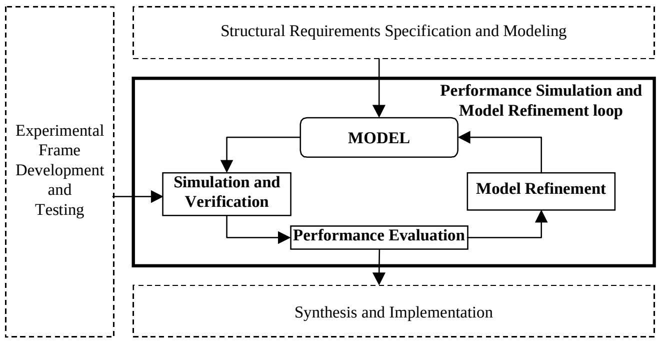
Structural Requirements Specification and Modeling
Performance Simulation and Model Refinement loop
Experimental Frame Development and Testing
MODEL
Simulation and Verification
Model Refinement
Performance Evaluation
Synthesis and Implementation

Figure 6 Performance Simulation and Model Refinement

A valid implementation is obtained if the performance model manages to fulfill the entire set of requirements. The model is refined to a state that supports conversion to a synthesizeable design description in the Synthesis and Implementation block.

2.5. Synthesis and implementation

The ability to synthesize descriptions that define a physical realization of the design is an important feature of our SONORA environment. These descriptions must accurately reflect the design model and should facilitate the fabrication of a system prototype. The descriptions

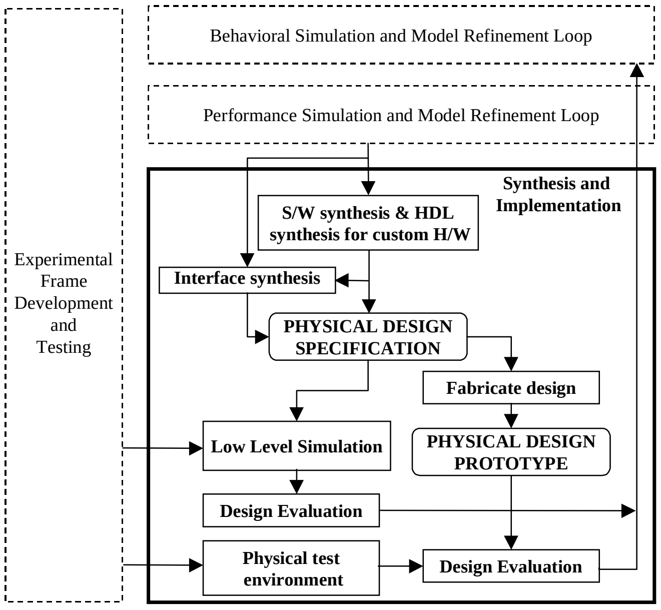
Behavioral Simulation and Model Refinement Loop
Performance Simulation and Model Refinement Loop
Synthesis and Implementation
S/W synthesis & HDL synthesis for custom H/W
Experimental Frame Development and Testing
Interface synthesis
PHYSICAL DESIGN SPECIFICATION
Fabricate design
Low Level Simulation
PHYSICAL DESIGN PROTOTYPE
Design Evaluation
Physical test environment
Design Evaluation

Figure 7 Synthesis and Implementation

will be made up of many parts to cover the various aspects of the design.

Given an assignment of functions to components from the model mapping process, the software and hardware descriptions of the mapped functions must be generated. Control architectures for the components must also be defined. The control architecture determines how the functions executing on a particular component communicate and are coordinated. The description must also define a system control architecture that defines how the components communicate and how they are coordinated. In addition to these control architectures, there is also the physical architecture for the system, i.e., its hardware components and interfaces. The hardware components have been defined by the model mapping but the interfaces must be defined during this phase in order to complete the physical design description.

There currently exist commercial tools that synthesize hardware and software descriptions from abstract atomic model components [14,18]. Syntheses of the remaining aspects are current areas of research. An approach for the synthesis of the hardware architecture, process mapping, and process scheduling has been presented by Yen and Wolf [30]. A system to perform synthesis of low-level interfaces of hardware components has been demonstrated by Chou et al. [31]. A focus of our research will be to explore methods to improve the synthesis of these design aspects.

An approach that will be explored in the SONORA environment will be the use of architectural patterns. These templates will define structures that occur often in the domain of embedded systems design. The results from model mapping and high level simulation and analysis will provide constraints for the synthesis process.

The physical design specification will feed both low level simulation and fabrication of a physical prototype. Through simulation and test both of these representations of the system will be evaluated based on the original requirements embodied in the experimental frames. The detailed simulation will precede fabrication and the results of which will be used as an input to the decision to proceed with fabrication. Evaluation of the physical prototype will either confirm that an acceptable design has been generated or that further refinement is necessary. It should be pointed out that detection of unsatisfied requirements at this late stage is intended to be extremely rare due to the emphasis on early simulation at the higher levels of the design process.

2.6. Experimental frame development and testing

Model testing is carried out through the use of experimental frames. The experimental frame concept is presented in detail in [7]. It is a construct that helps the model builder to keep the primary model (the object of interest) separate from the stimuli and responses used for evaluation during simulation.

Experimental frames are generated in parallel to the system modeling activities. The experimental frames are designed to test the system against the functional and behavioral requirements. Multiple experimental frames are generated to cover multiple aspects of testing. Experimental frames containing event scenarios will be used to test systems or portions of systems which are transaction oriented. These scenarios contain both system inputs and expected outputs and are generated from the functional and behavioral requirements specification. Experimental frames for aggregate functions such as throughput are adapted from existing frames in an experimental frame model base.

The set of experimental frames embodies the suite of tests for the system being designed. This suite shall be used to evaluate all potential designs. Maintaining a consistent suite of experimental frames facilitates the comparison of potential designs during trade-off analysis.

In addition to being used at the abstract modeling and simulation level, this suite of experimental frames shall be adapted and used to test the system at all

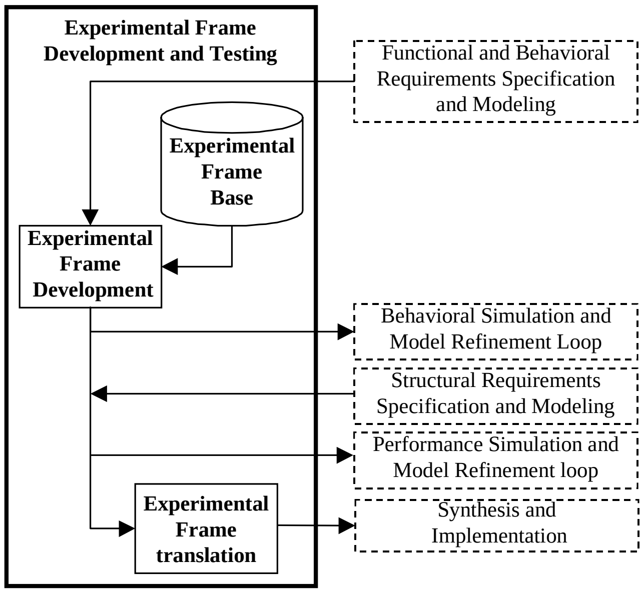
Experimental Frame Development and Testing
Functional and Behavioral Requirements Specification and Modeling
Experimental Frame Base
Experimental Frame Development
Behavioral Simulation and Model Refinement Loop
Structural Requirements Specification and Modeling
Performance Simulation and Model Refinement loop
Experimental Frame translation
Synthesis and Implementation

Figure 8 Experimental Frame Development and Testing

downstream levels of the design process. Translations will be made to testbenches for hardware level computer simulation and to a test environment that will enable us to test of a physical prototype.

Non-functional requirements and design choices shall be used as needed to translate the experimental frames into forms suitable for use at the lower levels. Maintaining a consistent set of tests based on the experimental frame suite will insure that all executable design representations, from the initial functional model to the physical realization, meet the system requirements.

2.7. Design management and tool support

One of the factors determining the usability of an environment is the method of tool integration. In our system, we plan to incorporate a design manager that will assist designers in controlling the flow of their activities by suggesting the choice of tools, interfacing with a revision control system, invoking tools and checking consistency of design data.

The communication between tools can be accomplished in a similar way to the FIELD environment [32]. There is a set of message interfaces for different types of tools, e.g. simulation, partitioning, compiling, which define the basic interface for that type of tool. To communicate with a commercial tool, an interface module can receive messages in the common format from the design manager, then convert the message into the format expected by the design tool. When a commercial tool performs more than one stage of the design process, the design manager has multiple interfaces with that tool.

3. Conclusions and future work

We are currently implementing SONORA on a network of UNIX workstations by integrating commercial and academic tools. We are planning to work with the graphical interfaces provided by commercial tools for design entry as well as informal text descriptions. Requirements can be entered prior to the modeling phase and updated during model refinement using the STATEMATE MAGNUM Requirements Tracer^{TM}. The requirements will be formatted in a semi-formal manner and extracted to allow custom tools to generate top level functional model structures and test cases for experimental frame construction.

Various modeling formalisms will be used in combination to be able to accurately reflect the different modeling aspects: DEVS, SES, and StateCharts to build a complete model specification. The resulting model will then be simulated with the appropriate simulation engines. We currently use DEVS-Java, the most recent implementation of the DEVS formalism [21], to simulate models of a system. However, we are investigating the automatic generation of DEVS models from StateChart [14] descriptions as well as the use of DEV and DESS (Difference Equation System Specification) [33] to represent system components that are more appropriately modeled in the continuous time domain.

Research is being conducted on introducing simulation results into a BBN during the simulation cycle. When the model is refined to a synthesizable level, the information from the BBN is used by model-to-realization mapping algorithms to prepare the verified model for prototype synthesis.

The use of the hardware and software language synthesis tools within STATEMATE [14] is finally considered for use in SONORA to provide the realization descriptions for functions (e.g., C and VHDL). Synthesis of control and communication descriptions requires further research in order to realize a fully automated prototype synthesis. The advantage of SONORA over other environments is that it will be able to heavily leverage from the benefits offered by model-based codesign.

Acknowledgments

This work has been supported by the National Science Foundation under grant No. 9554561 “Hardware/Software Codesign for High Performance Systems” in cooperation with Siemens AG, Central Research and Development Laboratories, Munich, Germany. We thank Mr. Joe Hanson at the ALTERA University Program for providing an ALTERA ISP Demo Board and the corresponding development software for our preliminary design studies.

4. References

[1] M. Chiodo, P. Giusto, A. Jurecska, H.C. Hsieh, A. Sangiovanni-Vincentelli, and L. Lavagno, “Hardware-Software Codesign of Embedded Systems,” *IEEE Micro*, 14(4), pp. 26-36, 1994.

[2] D. Gajski, S. Narayan, F. Vahid, and J. Gong, *Specification and Design of Embedded Systems*, Englewood Cliffs, NJ: Prentice-Hall, 1994.

[3] S. Kumar, *A Unified Representation for Hardware/ Software Codesign*, Ph.D. Dissertation, University of Virginia, 1995.

[4] G. De Micheli and R.K. Gupta, “Hardware/Software Co-Design,” *Proceedings of the IEEE*, 85(3), pp. 349-65, 1997.

[5] J.W. Rozenblit and K. Buchenrieder (Eds.), *Codesign: Computer-Aided Software/Hardware Engineering*, IEEE Press, 1994.

[6] S. Schulz, J.W. Rozenblit, M. Mrva, and K. Buchenrieder, "Model-Based Codesign," *IEEE Computer*, 31(8), 1998.

[7] J.W. Rozenblit, "Experimental Frame Specification Methodology for Hierarchical Simulation Modeling," *International Journal of General Systems*, 19(3), pp. 317-336, 1991.

[8] R. Ernst, J. Henkel, and T. Benner, "Hardware-Software Cosynthesis for Microcontrollers," *IEEE Design and Test of Computers*, 10(4), pp. 64-75, 1993.

[9] J. Bhat and F. Taku, "A seven-layer model of framework functionality," Electronic Engineering, pp. 67-73, September 1990.

[10] D. S. Harrison et al., "Electronic CAD Frameworks," Proceedings of the IEEE, v 78 n 2, pp. 393-417, February 1990.

[11] F. Bretschneider, *A Process Model for Design Flow Management and Planning,* PhD Dissertation. Department of Computer Science, University of Kaiserslautern, Germany, July 1992.

[12] ALTERA Corporation, *Max+PLUS II Getting Started,* 1996.

[13] Cadence Design Systems Inc., Concept: Mixed Level Design Entry System Product Description, 1997.

[14] D. Harel, "STATEMATE: A Working Environment for the Development of Complex Reactive Systems," *IEEE Transactions on Software Engineering,* 16(4), pp. 403-14, 1990.

[15] N.A. Markovitch and D.M. Profozich, "ARENA Software Tutorial," *Proceedings of the 1996 Winter Simulation Conference*, San Diego, pp. 437-40. 1996.

[16] Mentor Graphics Corporation, *Seamless CVE ™ Product Description,* 1996.

[17] A. Mullarney, "MODSIM III —A Tutorial," in *Procroceedings of the 1996 Winter Simulation Conference*, San Diego, pp. 542-46, 1996.

[18] Nu-Thena Systems Inc., Foresight: A Product Family Overview, 1994.

[19] Xilinx Inc., Development Systems Product Overview, 1996.

[20] C. Ebert, "Experiences with Colored Predicate-Transition Nets for Specifying and Prototyping Embedded Systems," *IEEE Transactions on Systems, Man, and Cybernetics – Part B: Cybernetics*, 28(5), October 1998.

[21] B.P. Zeigler, *Multifaceted Modeling and Discrete Event Simulation,* Academic Press, London; Orlando, 1984.

[22] J. Pearl, Probabilistic Reasoning in Intelligent Systems: Networks of Plausible Inference, Morgan Kaufmann Publishers, San Mateo, CA, 1988.

[23] E. Charniak, "Bayesian networks without tears," AI Magazine, Vol. 12, No. 4, pp. 50-63, winter 1991.

[24] L. C. Van der Gaag, "Bayesian Belief Networks: Odds and Ends," Computer Journal, Vol. 39, No. 2, pp. 97-113, 1996.

[25] J. T. Olson, J. W. Rozenblit, "Framework For Hardware/Software Partitioning Utilizing Bayesian Belief Networks," *IEEE Proceedings of the International Conference on Systems, Man and Cybernetics*, October 1998.

[26] F. Vahid, D. Gajski, "SLIF: A Specification-Level Intermediate Format for System Design," *Proceedings of the ED&TC, European Design and Test Conference*, pp. 185-189, 1995.

[27] J. Rumbaugh et al, *Object-Oriented Modeling and Design*. Prentice Hall, 1991.

[28] J.W. Rozenblit and J. Hu, "Integrated Knowledge Representation Management in Simulation Based Design Generation," *IMACS Journal of Mathematics and Computers in Simulation*, pp. 34(3-4), 262-82, 1993.

[29] J.W. Rozenblit and Y.M. Huang, "Rule-Based Generation of Model Structures in Multifaceted Modeling and System Design," *ORSA Journal on Computing,* 3(4), pp. 330-44, 1991.

[30] T.-Y. Yen and W. Wolf, Hardware-Software Co-Synthesis of Distributed Embedded Systems, Kluwer, Boston, 1996.

[31] P. Chou, R. Ortega, and G. Borriello, "Synthesis of the Hardware/Software Interface in Microcontroller-Based Systems," International Workshop on Hardware/Software Codesign, Estes Park, Colorado, September 1992.

[32] S. Reiss, The Field Programming Environment: A Friendly Integrated Environment for Learning and Development, Kluwer Academic Publishers, 1995.

[33] H. Praehofer, "System Theoretic Formalisms for Combined Discrete-Continuous System Simulation," *International Journal of Systems*, V. 19, pp. 219-40, 1991.