Chapter 7

Systems Design: A Simulation Modeling Framework

J.W. Rozenblit

This chapter surveys a model-based approach to the design of complex, multifaceted systems. The discussion is organized as follows: first, a brief introduction to the model-based design (MBD) methodology is given. Then, supporting concepts, knowledge representation and modeling techniques are discussed. Brief, illustrative examples are given to elucidate the theory-based notions. The chapter concludes with postulates for a computer-aided system that can automate the design of heterogeneous systems.

7.1 Introduction and Motivation

Modern engineering design is a highly complex process. It involves a multiplicity of objectives, constraints, materials, and configurations. Despite great strides in computational tools such as high performance workstations, distributed and concurrent processing environments intended to help to cope with this rising complexity, the design process remains error prone. Given the often severe constraints imposed by cost, environmental impacts, safety regulations, etc., designers are forced to make compromises that would not be necessary in an ideal world

In the late nineteen eighties and early nineteen nineties, we had witnessed a proliferation of efforts to support the design process with computer aided tools and environments. Most successful in that undertaking was the electronic design automation community whose efforts led to the definition of "electronic design framework" and subsequent developments of design environments and tools such as Falcon and Synopsis.

The term framework in electronic design automation denotes a computer-based, integrated design environment that binds and supports design tools [2]. In their seminal paper, Harrison et al. [8] defined a framework as "...all the

H. S. Sarjoughian et al. (eds.), *Discrete Event Modeling and Simulation Technologies*
© Springer Science+Business Media New York 2001

underlying facilities provided to the CAD tool developer, the CAD system integrator, and the end user necessary to facilitate their tasks."

The CAD Framework Initiative (CFI) viewed a framework as a collection of extensible programs/modules used to develop a unified CAD system [2] For the sake of brevity, we do not discuss the history of electronic CAD here nor do we trace the evolution of the framework concept. (Excellent summaries are given in [3,8,23].

Another notable thrust in the efforts to support complex design was the Defense Advanced Research Projects Agency (DARPA) program in concurrent engineering. In 1987, DARPA had asked experts from industry, academia, and government to investigate the shortcomings in the defense-related product development process. Japanese product development techniques were studied in which the so-called "tiger approach" was applied. In this approach, all personnel involved in the project were assembled at its conception in order to find a solution which addresses the participants' individual concerns but also benefits the team as a whole. From this relatively general description, stemmed the term "concurrent engineering" (CE). The term was refined by Winner et al. in [27] as "...a systematic approach to the integrated, concurrent design of products and their related processes, including manufacture and support. [CE] is intended to cause the developer, from the outset, to consider all elements of the product life cycle from conception through disposal, including quality, cost, schedule, and user requirements."

To transplant the approach into the American research and development environment, DARPA concluded that an advanced computer technology was needed to create virtual teams who might be scattered all over the country. Thus came about the DICE program (DARPA Initiative in Concurrent Engineering) [24]. Special issues of archival journals had reported extensively on CE [9,10].

Simulation modeling had long been recognized as a useful tool in assessing the quality of sub-optimal design choices and arriving at acceptable trade-offs prior to the physical realization of the system being designed. Often termed "virtual prototyping", or "simulation-based design" (SBD), this approach was endorsed by the concurrent engineering community and applied primarily in the mechanical engineering domain [5]. Subsequent, efforts focused on the development of techniques that would integrate distributed simulation technologies, physics-based modeling, virtual and collaborative environments. The long-range goal of such an integration is the application of these, and other, technologies that permit integrated process and product development (IPPD). The perceived benefits of simulation-based design clearly justify the research and development expenditures. If successfully applied to complex engineering enterprises, SBD can facilitate assessment of products and process designs early in the lifecycle and eliminate the cost of physical prototyping.

Our position is that computer simulation and other advanced computational tools are of limited effectiveness without a methodology to induce a systematic handling of the multitude of goals and constraints impinging on the design process. Therefore, our work focuses on the development of techniques in which

design models can be synthesized and tested by simulation within a number of objectives, taken individually or in trade-off combinations. We strive to provide a uniform treatment of the design process at different levels of system representation and abstraction. By providing a spectrum of performance evaluation methods, including trade-off measurements and evaluation of multilevel, multicomponent, hierarchically specified models, our approach facilitates description of designs through quantitative and, more generally, comparative measures.

Our approach, termed model-based design (MBD) [16,17,18,19], was initially strongly influenced by multifaceted modeling concepts developed by Zeigler [29] in the late seventies and mid-eighties. The approach is primarily intended to support the development of simulateable models of the system under design. It offers: a) representation methods to capture structural and behavioral design knowledge, b) heuristics for managing design space complexity, c) techniques for simulation-based design assessment and trade-offs, and d) methods for design partitioning [13].

This chapter surveys the fundamental tenets of the MBD framework. A high level example serves to illustrate the methodology and its various phases. For an comprehensive exposition of the formal concepts underlying the methodology and its phases, we refer the reader to [16,178,18,19].

7.1.1 System Design and Modeling: Synergies

As opposed to system analysis where models are typically derived from an existing real system, in design the model comes first as a set of "blueprints" from which the system will be built, implemented or deployed. The blueprints take several forms; they could be simple verbal descriptions, a set of equations, an architectural drawing, or a netlist file. The goal of such defined systems design is to study models of design before they are physically realized.

Here, a question arises: "How can modeling and simulation concepts support systems design?" To address this question, we enumerate some principal elements in the dynamics of the design process and relate them to simulation modeling.

1. Designs are created by individuals who use basic problem solving techniques such as problem definition, proposal of a solution, and test of the solution against the problem statement and requirements. Modeling, as a creative act, follows similar steps. To build a model, the modeler interprets requirements and objectives of the project, proposes a suite of simulateable models that serve as "design blueprints." Simulation is a means of executing the models and testing how well they meet the project's requirements.
2. Designs are often of a large scale. Thus, techniques are needed for decomposing the design problem into subproblems that are easily

comprehensible to the designers. Partial solutions could then be generated and integrated using proper aggregation techniques. Simulation modeling methodologies provide techniques for model decomposition, hierarchical specification and aggregation of model components.

3. The attributes of design are described in the form that allows comparative studies and trade-off analyses. Simulation-based design performance evaluation uses model generated data for such analyses.
4. Complex designs comprise heterogeneous components – techniques are needed to properly integrate them into a complete system. Recent advances in embedded systems modeling [1,4,6,11,13] and real-time systems modeling are a promising step in this direction.

In what follows, we survey the MBD framework and illustrate its elements on an example.

7.2 The MDB Methodology

Our approach to design is shown in Figure 7.1. The process depicted in the figure uses simulateable models as virtual prototypes of the system under design (SUD). The core activities in the framework are system specification, modeling, simulation, refinement, and partitioning (technology assignment).

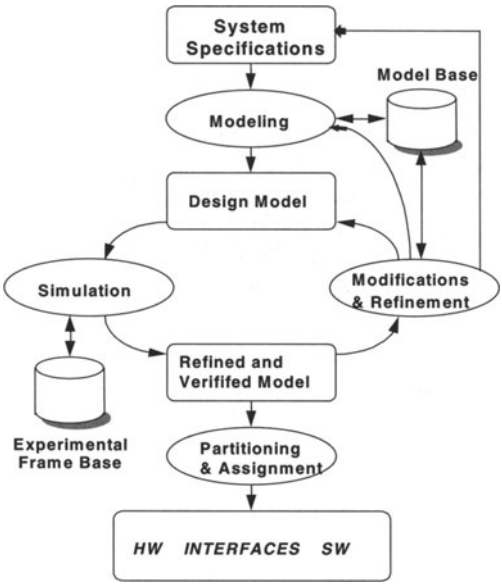

Figure 7.1: Simulation-Based Design Methodology

To build a simulateable design model, we first construct object models, ie. representations of components from which a system will be built, their relationships, and attributes. Such models are given behavioral specification so that they can be simulated.

Simulation is then used in a two-fold manner: a) as a means of verifying the functionality of the proposed solution, ie. the execution of the model's dynamics to ensure that the behaviors are consistent with those perceived for the system being designed, and b) as a way of assessing how well performance requirements are met by the proposed solution (for example, we can collect data on component utilization, system's throughput, etc.).

An assessment of simulation data leads to either further model refinement and design modifications, or to the system partitioning and technology assignment phase. During this phase, decisions as to which components are realized in hardware, software and interfaces are made. (This particular phase is pertinent to design of heterogeneous systems that contain such mixed elements).

Our overall design model combines various representations, simulation modeling, and heuristic techniques. They are now described in more detail.

7.3 MBD Process Elements

7.3.1 System Specifications

In the first phase, the designer converts the system's requirements and constraints into a formal specification. This specification defines the interface between the system and its environment and the system's functionality. Nonfunctional requirements such as size, weight, etc. are also documented. Since our approach strives for implementation independence, designers can refine the specification without modifying any physically realized components.

7.3.2 Modeling

This is the core activity that leads to virtual design prototypes. In our approach, we take a holistic view that aims at the development of the overall system's model rather than its individual components. A model is a set of instructions for generating data. Valid model-generated data is a subset of the system's behavioral data. A specification of the system and its environment forms the basis for building models that correspond to a set of questions about the design, including its objectives and reason for being [15,17]. The modeling phase of MBD is depicted in Figure 7.2.

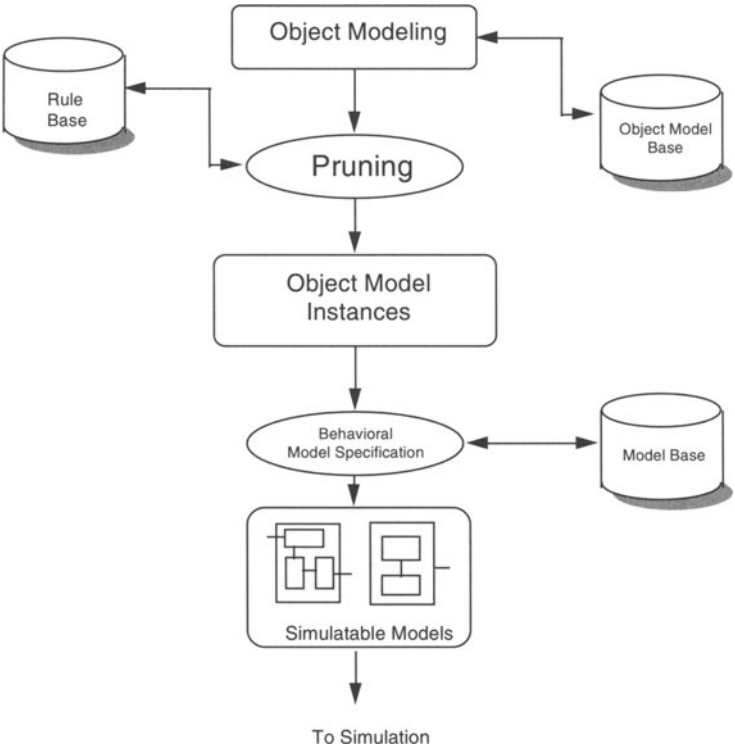

Figure 7.2: Generation of Simulateable Models

7.3.2.1 Object Modeling

The design model construction process begins with developing a representation of design components, their decompositions, variants, and attributes. As a step toward a complete knowledge representation scheme for design support, we combined the decomposition, taxonomic, and coupling relationships in a knowledge representation scheme called the *system entity structure* (SES). Previous work [16,17,18,19] identified the need for representing the structure and behavior of systems, in a declarative scheme related to frame-theoretic and object-based formalisms. The elements represented are motivated, on the one hand, by systems theory concepts of decomposition (ie. how a system is hierarchically broken down into components) and coupling (ie. how these components may be interconnected to reconstitute the original system). On the

other hand, systems theory has not focused on taxonomic relations, as represented for example in frame-hierarchy knowledge representation schemes. In the SES scheme, such representation concerns the admissible variants of components in decompositions and the further specializations of such variants.

The interaction of decomposition, coupling and taxonomic relations in an SES affords a compact specification of a family of models for a given domain. In a system entity structure, entities refer to conceptual components of reality for which models may reside in a model base. Also associated with entities are slots for attribute knowledge representation. An entity may have several aspects, each denoting a decomposition, and therefore having several entities. An entity may also have several specializations, each representing a classification of possible variants of the entity.

Object modeling requires several types of decisions on the part of design project engineers. The classical object model development process advocated by Rumbaugh et. al. [20] involves a syntactic analysis of the requirements document. Through this analysis, noun, verb, and adjective phrases are identified. Classes are then constructed to reflect the nouns (objects that represent system components). Association links are derived from the verb phrases. For example, a "has part" phrase can be directly expressed through a decomposition link while an "is a kind of" phrase can be reflected through the generalization relation.

The syntactic analysis results in the initial object model. However, the model's refinement is a responsibility of the modelers who make the following decisions:

a) further identify how components decompose into subcomponents in the project's domain and provide a set of alternative decompositions,
b) identify sets of variants for components specified in various decompositions (for example, a computer display type could be an LCD or a CRT monochrome, or a CRT color display),
c) identify attributes for the components that describe the components' properties and characteristics salient to the project at hand.

The system entity structure organizes possibilities for a variety of system decompositions and taxonomies and, consequently, a variety of model constructions. Its generative capability facilitates convenient definition and representation of models and their attributes at multiple levels of aggregation and abstraction. More complete discussions of the system entity structure and its associated structure transformations are presented in [16,17]. The SES representation can be rendered using the Object Modeling Technique (OMT) notation (a less complex precursor to the Unified Modeling Language). Here, we illustrate its expressive power using the basic concepts from an Air Traffic Display/Collision Warning (and Monitoring) System (ATD/CWS) system design example.

The ATD/CWM system is intended to monitor air traffic and issue warnings should a threat of collision between a host aircraft and other aircraft occur.

Figure 7.3 depicts the system entity structure of the ATD/CWS system rendered using the OMT notation. The ATD/CWS is a part of a larger distributed network. Through the aggregation symbol (diamond in Figure 7.3) we show its decomposition into a) location records library, b) collision warning monitor, c) external interfaces (in OMT, the filled circle denotes one or more elements; thus the plural interpretation of the class object "External Interface"), and d) CPUs. The external interface object is classified through the specialization relation (or, as termed in the OMT notation, through the generalization relation depicted by the triangle symbol) into *radar, communication system, user interface, alarm device,* and *navigation system.* Furthermore, the radar object is specialized into *simple,* and *complex* radar. The air traffic display has two variants: a) color display and b) monochrome display.

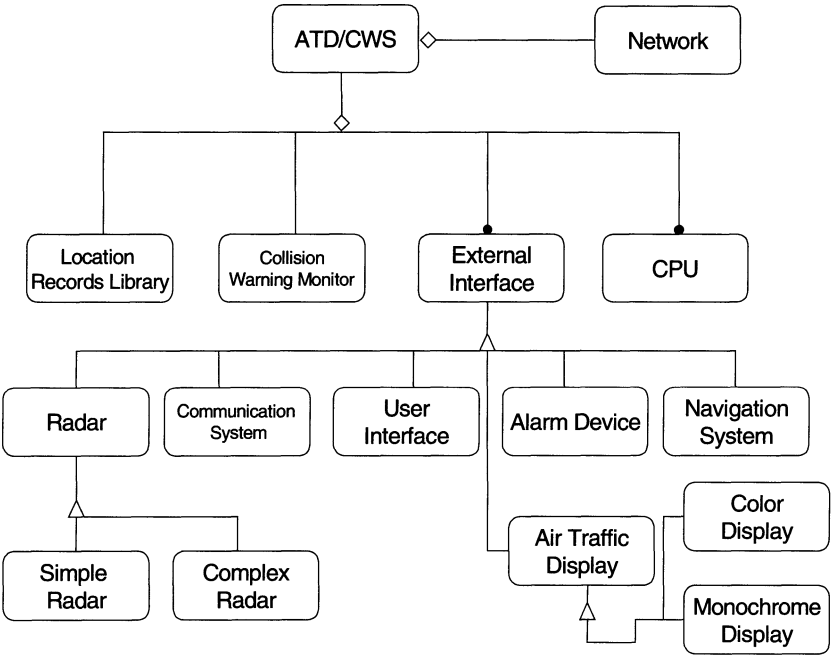

Figure 7.3: Object Model of the ATD/CWM System

The object model of Figure 7.3 can be refined further to show attributes of the system's components and their methods. In Figure 7.4, we refine the object instance of ATD/CWM to include the attribute *mode* (with a possible range *normal,* and *degraded*) and a method *detect collision.* This method would implement the collision detection algorithm. Similarly, we have refined the class object Location Records Library to show the structure of the Location Record object. Such an object has attributes that relay the flight information as shown in

Figure 7.5. In Figure 7.6, we illustrate a refinement of the Air Traffic Display and the Aircraft ID Icons.

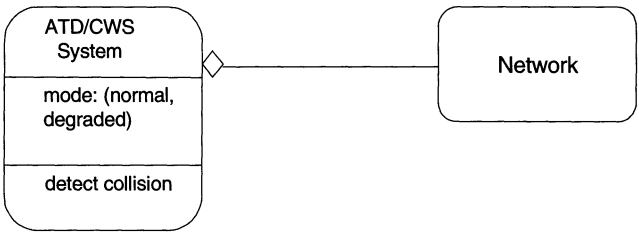

Figure 7.4: Refined Class Object ATD/CWM

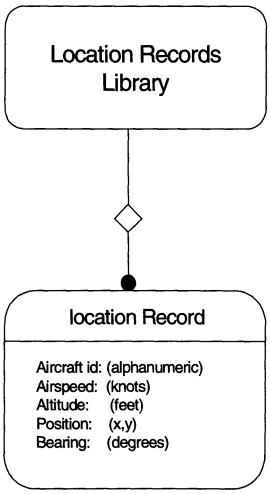

Figure 7.5: Refined Instance of Location Records Library Class

This high level representation, shown in the OMT diagrams, sets up an object model from which a specific design instance of system components (as well as their relationships) for the ATD/CWM system can be synthesized. We discuss this synthesis in the following section.

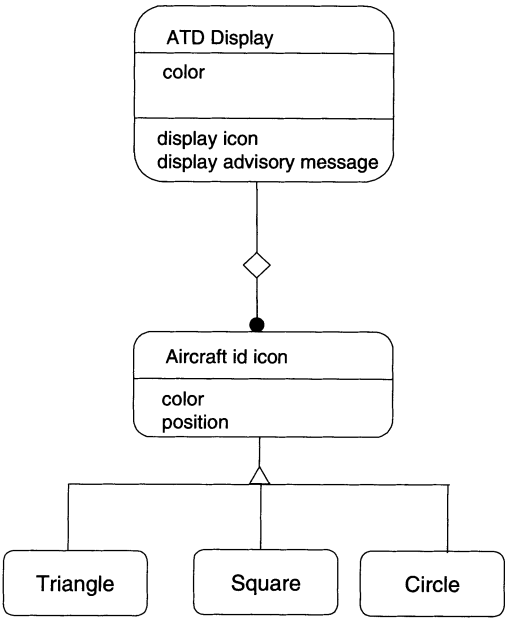

Figure 7.6: Refinement of the Air Traffic Display Object

7.3.2.2 Pruning

In our methodology, a model is synthesized from elements stored in the model base. More specifically, behaviors are associated with design components identified in the object models. This synthesis is the result of *pruning* a substructure from the system entity structure or an OMT diagram. Pruning can be viewed as a knowledge-based search through the space of candidate solutions to the design problem. This is consistent with a commonly taken view that design is a search process in which a satisfactory design solution is produced from a number of alternatives [17,19]. Those alternatives come from knowledge of the relevant domain. We use *production rules* [19] to represent design objectives, constraints, requirements and performance expectations. The aim of pruning is to recommend plausible object model instances for further design assessment using simulation.

The following steps are required to provide the rules that guide pruning of the system entity structure:

1. for each specialization (ie. generalization relationship), specify a set of rules for selecting an object; that is select an instance of a system's component from possible variants,
2. for an entity with several aspects, ie. decompositions (or in the OMT terminology aggregations), specify rules for generating a unique aspect;
3. for each aspect, ie. aggregation, specify rules that ensure that the objects selected from specializations are configurable, ie., the components they represent can be validly coupled.

Thus, to guide the search through the design space defined by the object models, we specify a) selection rules for choosing a component instance from a variety of elements offered by the generalization relations, b) synthesis rules for combining the selected instances.

To specify the pruning rules, the knowledge engineering team elicits domain expertise from subject matter experts (SMEs). The expertise is encoded in the form of if-then production rules that constrain the choices offered by specializations and provide a coupling recipe for aggregating components identified in decompositions. In addition to encoding the experts' domain knowledge, the rules may translate a requirements statement into an operational means of selecting the most adequate choice of object models for the system under design (SUD) (for example, when distributed processing is required, select a multiprocessor computing platform).

Several attributes play a role in this phase of the SBD process:

a) in the elicitation process, subject matter experts provide knowledge that is biased by personal experiences and preference,
b) communication skills of the elicitors, ie. knowledge engineers, as well as SMEs impact the confidence in information provided; thus, confidence factors are typically embedded in the rules (they are based on the assessment of how reliable the information provided by experts is),
c) the pruning methodology may be driven by designers' preferences that stem from their experience and intuition.

We now illustrate the pruning concepts by using the ATD/CWM example. Recall the object model of Figure 7.3. Through the specialization relations (captured in OMT using the triangle symbol) we outline the choices for component selections. More specifically, a simple or a complex radar can be selected for the radar components. A choice of a color or a monochrome display is given for the air traffic display device. Moreover, through the one-to-many relation (depicted in OMT by the filled circle), the ATD/CWM system may have more than one CPU.

The following production rules can be used in the selection and synthesis of a specific instance of the system.

Rules for Radar Selection

```
If required aircraft_bearing is two-dimensional and
   range is <= 250 miles
   then radar_type is simple

If required aircraft_bearing is three-dimensional and
   range is > 250 miles
   then radar_type is complex
```

Rules for Display Selection

```
If aircraft type classification based on color separation
   then display_type is color

If aircraft type classification based on brilliance and
shape separation
   then display_type is monochrome
```

Synthesis Rule for ATD/CWM

```
If selected radar_type is simple and
   selected display_type is color and
   required emergency mode is degraded
   then the ATD/CWM system is configurable and
        it has at least 2 CPUs and
        it has a simple radar  and
        it has a color display and
        it has a location records library and
        it has a collision warning monitor and
        it has a communication system and
        it has an alarm device and
        it has an navigation system and
        it has a user interface.
```

In our methodology, the selection and synthesis rules (given in the form of production rules) are used by a reasoning engine that generates a specific design instance given specific values of design variables [19]. Such values instantiate design parameters which are associated as attributes with various entities (object classes). For example, a design requirement may impose a choice of two-dimensional

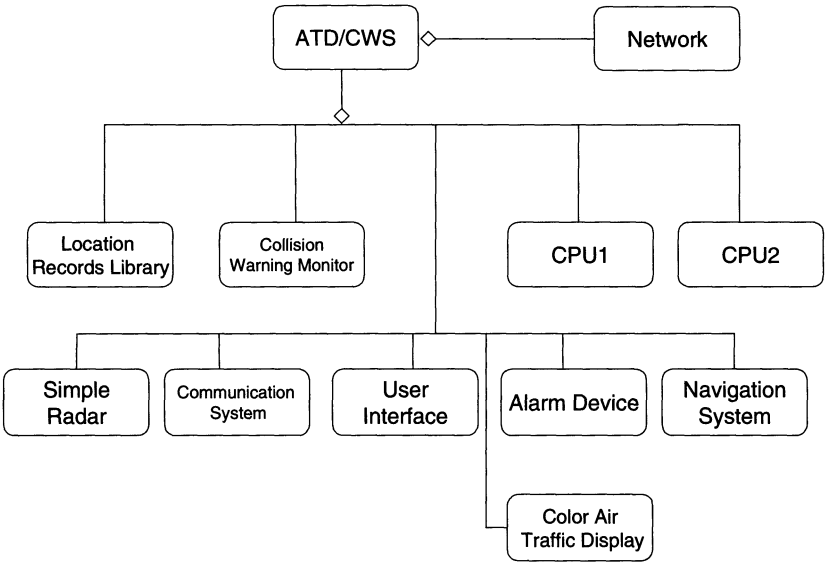

Figure 7.7: Pruned Instance of the ATD/CWM Object Model

aircraft bearing or a color-based identification of aircraft type on the air traffic display. Those values are entered by the designer during the pruning process. Their combinations result in a specific object model instance that results from the family of design configurations given by the overall system's object model. In Figure 7.7, we show a possible instance of the ATD/CWM that can be pruned from the system entity structure object model (Figure 7.3).

7.3.2.3 Behavioral Model Specification

Behaviors can be associated with object models instances selected through the pruning process using Petri nets, StateCharts [7], Discrete Event System Specification (DEVS) [29], other finite state machine based formalisms, or continuous system specifications. In MBD, we strive to use formal modeling methods that allow the designer to construct a hierarchy of models using representations that are closed under coupling. The type of specification language used in modeling is very important. The specification must accommodate different levels of granularity so that the developer can map components at different levels of abstraction to corresponding hardware or software modules.

In our framework, we have worked extensively with the Discrete Event System Specification formalism [29]. DEVS is a general, formal specification language that allows the designers to specify a system as a mathematical object. The specification comprises a time base, inputs, states, and outputs and functions of determining next states and outputs. Designers make no decisions about how to build the components at this stage; they connect elementary blocks hierarchically until they arrive at a preliminary model that conforms to the project's requirements. We have used DEVS and its implementation environment DEVSJAVA to build models in a hierarchical and modular fashion [21]. This manner of model development permits a systems-oriented approach that has not been possible in the past with such simulation languages as Simscript or SIMAN. We must note, however, modularity and module composition features are now common in object-oriented simulation languages such as MODSIM or Modula.

7.3.3 Simulation

In this phase, execution of design models is carried out using the *experimental frame* paradigm [10,12,17]. Experimental frames define conditions under which models can be observed and experimented with. An experimental frame reflects modeling objectives. The statement of objectives is translated into specific performance measures. Necessary output, input and control variables are defined so that such measures can be obtained through simulation experiments. An experimental frame plays the following roles: a) it subjects a model to input stimuli (which represent potential interventions into the model's operation), b) it observes the model's reactions to the input stimuli and collects the data about such reactions (output data), and c) it controls the experimentation by placing relevant constraints on values of the designated model state variables and by monitoring these constraints.

Figure 7.8 illustrates the separation of a model and its experimental frame. A model can be executed in various experimental frames, each reflecting a specific

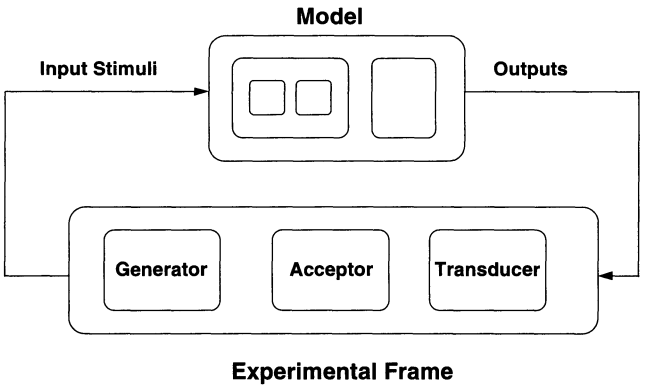

Figure 7.8: Model/Experimental Frame Coupling

objective of a simulation study. An experimental frame can be coupled to several design models (each reflecting an alternative, virtual design solution) so that trade-offs can be made and the best model can be selected with respect to the specific assessment objective that this frame represents.

Therefore, the experimental frames provide the design team with a quantifiable means of trading off design solutions (in the form of simulation models) with respect to the set of design objectives. Such trade-off decisions can be made by team members using multiple criteria decision making (MCDM) techniques as depicted in Figure 7.9. In the scenario shown in the figure, alternative design solutions are given as models $M_1,\ldots,M_n$. The experimental frames, EF_1 and EF_2 reflect two performance objectives, eg. component utilization (EF_1) and task throughput (EF_2). The models can now be cross evaluated in both frames and the tradeoff solution (the model M*) can be selected.

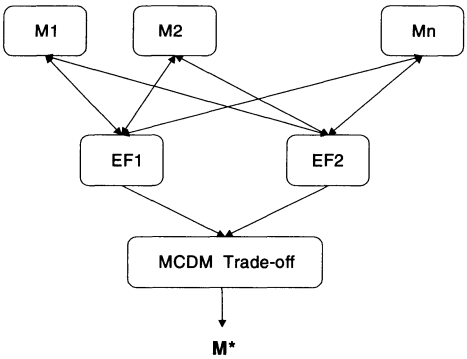

Figure 7.9: Trading-off Design Solutions using Experimental Frames and MCDM

Experimental frames are given concrete form – employing the concepts of automata theory, a frame can be defined as a composition of a *generator* which produces the input segments sent to a model, an *acceptor*, ie., a device that continually monitors the simulation run for satisfaction of constraints, and a *transducer* which collects the input/output data and computes summary performance measures. Experimental frame template specifications can be stored in the experimental frame base (recall Figure 7.1) for reuse and rapid simulation run setup.

We refer again to the ATD/CWM system example to illustrate the experimental frame concept. Assume that one of the objectives of the simulation based design process is to determine the optimal number of the CPUs in the system so that their utilization is maximized. Gaining introspection into this aspect of the system's performance would assist designers in selecting an appropriate number of processors in the air traffic detection system.

First, we translate the utilization objective into a specific performance measure. Utilization is measured by monitoring the busy/idle ratio of the CPUs' operation. In addition, we introduce a variable that computes the joint utilization of the system's processors so that we can determine the fraction of time that both processors are simultaneously busy (the "and" is a logical conjunction in the formula below that we use for computing the joint utilization). We now specify the requisite elements of an experimental frame derived from these desiderata.

7.3.4 Experimental Frame CPUs Utilization

7.3.4.1 Generator

Generate input segments:

- radar sweep once per one-quarter second
- communication messages (a sequence of randomly distributed events)
- operator messages (a sequence of randomly distributed events)

7.3.4.2 Transducer

Monitor model variables:

CPU1.Status (with range: busy, idle, fail)
CPU2.Status (with range: busy, idle, fail)
Time (with range: non-negative reals)

Compute measures:

CPU1.Utilization = Time.CPU1.Busy / Global.Observation.Time
CPU2.Utilization = Time.CPU2.Busy / Global.Observation.Time

CPU.Joint.Utilization = Time (CPU1.Busy and CPU2.Busy) /
Global.Observation.Time

7.3.4.3 Acceptor

Monitor simulation run

Run until CPU1.Status = Fail or CPU2.Status = Fail or
Global.Observation.Time >= End.Time

Simulation (driven by a set of experimental frames) is followed by analysis of how well design functionality and performance are met. Based on this analysis, the design model at hand may be further refined and modified.

7.3.5 Design Modifications and Refinement

The ability to modify and refine designs in an iterative manner at the level of virtual prototypes is not well addressed in the systems engineering community. The development of design modification procedures and thereby the incorporation of simulation into a design feedback and iteration loop is an open, general research challenge.

A simple case would be to define methods to identify which elements and components of the design fail to meet the specification and provide this information to designers. The designers would then change the requirements or

refine the design models including their behavior, and would repeat the assessment using simulation.

A complex approach would incorporate reasoning and heuristic procedures that would automatically iterate through the design space by changing various parameters and would arrive at the optimal solution. Such procedures are possible for well-defined domains (for instance see [22] for examples in the VLSI interconnect design domain). However, for projects that do not afford a high degree of design automation, the refinement heuristics would be driven by attributes such as the experience of design engineers, confidence in validity of simulation models, ability of team members to reach a consensus, and change risk tolerance.

7.3.6 Partitioning and Technology Assignment

After the system design is completed and evaluated at the virtual model level, a physical realization of the system must be carried out. The translation from the model to the actual implementation is typically done based on the available technology constraints and performance estimates for realizations of components in software or hardware. In traditional design, the partitioning scheme is often tied to the target architecture. In our approach, mapping model components to hardware and software is not as limited since the design is independent of the implementation until this phase, which is relatively late in the design.

There is a great body of partitioning work that is well documented in the literature [1,12,26]. [1] provides an excellent review of the major classes of partitioning algorithms that not only can be used for VLSI circuit design, but for any system in which components are grouped and whose inter-group communication must be kept to a minimum. Among these classes of partitioning algorithms are the following: (1) move-based approaches such as greedy and iterative exchange algorithms, (2) geometric approaches such as vector partitioning, (3) combinatorial approaches such as max-flow min-cut, and (4) clustering-based approaches [1,26]. In addition, other researchers have either augmented the general algorithms (for example, Vahid [26] modified the min-cut algorithm for functional partitioning), or introduced new types of algorithms (such as Wolf who employed an object-oriented approach [28]).

In the design of heterogeneous systems, the choice of how to implement the system architecture can make significant differences in performance and reliability. In the past, a hardware platform was often chosen and then software was written for correcting the inadequacies of the hardware. Currently, however, research has progressed from the idea of partitioning hardware elements, to that of partitioning a high level functional model of a system. Figure 7.10 shows an example partitioning into hardware and software, with the system model containing four functional components (A, B, C, D) that are partitioned into hardware (A, C, D) and software (B).

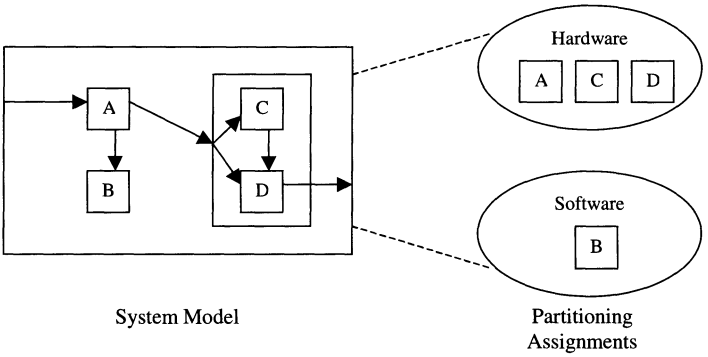

Figure 7.10: An illustration of the partitioning problem.

We have been exploring the application of Bayesian Belief Networks (BBN) [14] to the partitioning problem. The application of BBN to hardware/software partitioning was first introduced by Olson and Rozenblit [13]. Before classification into hardware or software can begin, a functional description of the model is created (in a manner similar to the Specification Level Intermediate Format (SLIF) [25]). 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.

Results are obtained from simulation and converted into evidence that is propagated throughout the 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. 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.

7.4 Towards an Integrated Design Support Environment

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 few successfully implemented environments have emerged that support heterogeneous embedded systems design. At the University of Arizona, we are developing a computer-aided design environment called SONORA [4]. 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.

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™. 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, to simulate models of a system. However, we are investigating the automatic generation of DEVS models from StateChart descriptions as well as the use of DEV and DESS (Difference Equation System Specification) 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 [7] 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.

7.5 Summary

To gather the strands up, we now summarize the steps and phases in the MBD process. The phases required to execute the methodology are:

- Decompositions, specializations, and attributes of components of the system being designed are conceptualized using the object modeling approach. We utilize the object model base as a repository of previous design modeling experience. Thus, we may retrieve an object model from this base that is applicable to the modeling domain at hand. Such a model is modified and enhanced with entities required in the new project.
- A rule base to be used in the pruning process is developed. Requirements and constraints are translated into production rules.
- Pruning is invoked which generates recommendations for candidate solutions to the design problem in the form of object model instances.
- Behaviors are associated with the object model instances.
- Relevant experimental frames that reflect design objectives are defined.
- Simulation is run and results are evaluated and design models are ranked. They are assessed and refined until a final design model is obtained. The above phases may be iterated in a feedback process.
- The final model (ie. virtual design prototype) is handed over to partitioning and technology assignment.

Acknowledgements

The material presented here is based in part on a report to Systems World, Inc. I appreciate the assistance of Dr. Stephanie White for facilitating this publication. I am also grateful to my research team members: Tony Ewing, John Olson, Steve Cunning and Stephan Schulz for all their contributions to the development of the hardware/software codesign framework.

References

[1] C. J. Alpert and A. B. Kahng, "Recent Directions In Netlist Partitioning: a Survey," Integration, the VLSI Journal, Vol. 19, No. 1-2, August 1995, pp. 1-81.

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

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

[4] S. Cunning, T.C. Ewing, J.T. Olson, J.W. Rozenblit, and S. Schulz, Towards an Integrated, Model-Based Codesign Environment. *Proceedings of the 1998 IEEE Conference on Engineering of Computer-Based Systems,* 136-143, Nashville, March 1999.

[5] M. Cutkosky et al. PACT: An Experiment in Integrating Concurrent Engineering Systems. IEEE Computer, January 1993, pp. 28-37.

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

[7] D. Harel et al. Statemate: A Working Environment for the Development of Complex Reactive Systems. IEEE Trans. on Software Engineering, 16(4), 403-414, April 1990.

[8] D. S. Harrison et al. Electronic CAD Frameworks. *Proceedingsof the IEEE*, v 78 n 2, FEB 1990, pp. 393-417.

[9] IEEE Computer. January 1993.

[10] IEEE Spectrum. July 1991.

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

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

[13] J.T. Olson and J.W. Rozenblit, A Framework for Hardware/Software Partitioning Utilizing Bayesian Belief Networks, *Proceedings of the 1998 IEEE Conference on Systems, Man and Cybernetics,* 3983-3988. San Diego, October, 1998.

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

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

[16] J.W. Rozenblit and B.P. Zeigler, Design and Modeling Concepts, In *International Encyclopedia of Robotics.* (Ed. R. Dorf), 308-322, John Wiley and Sons, New York, 1988.

[17] 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.

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

[19] 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), 330-344.

[20] J. Rumbaugh et al., Object Oriented Modeling and Analysis. Prentice Hall, 1991.

[21] S. Schulz, J.W. Rozenblit, M. Mrva, and K. Buchenrieder, "Model-Based Codesign: the Framework and its Application, *IEEE Computer,* August 1998.

[22] T. Simunic, J.W. Rozenblit, and J. Brews, VLSI Interconnect Design Automation Using Quantitative and Symbolic Techniques, *IEEE Transactions on Components, Packaging, and Manufacturing Technology,* 19(4), 803-812, Nov. 1996.

[23] Special Technology Area Review (STAR) on Computer Aided Design.Report of the Department of Defense Advisory Group on ElectronDevices, Washington, D.C., (open publication), Feb. 1993.

[24] R. A. Sprague, K. J. Singh, and R. T. Wood. Concurrent Engineering in Product Development. {\em Ieee design and test of computers}, MAR 01 1991 v 8 n 1, pp. 6-13.

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

[26] F. Vahid, "Modifying Min-Cut for Hardware and Software Functional Partitioning," Proceedings of the Fifth International Workshop on Hardware/Software Codesign. CODES/CASHE '97, pp. 43-48.

[27] R. Winner et al. The Role of Concurrent Engineering in Weapons Acquisition, IDA Report R388, Institute of Defense Analysis, Washington, D.C., 1988.

[28] W. Wolf, "Object-Oriented Cosynthesis of Distributed Embedded Systems, ACM Transactions on Design Automation of Electronic Systems, Vol. 1, No. 3, July 1996, pp. 301-31.

[29] B.P. Zeigler, Multifaceted Modeling and Discrete Event Simulation. Academic Press, 1984.