Proceedings of the 1988 Winter Simulation Conference
M. Abrams, P. Haigh, and J. Comfort (eds.)

Towards an implementation of a knowledge-based system design and simulation environment

Jerzy W. Rozenblit, Tag Gon Kim, and Bernard P. Zeigler

Department of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721

ABSTRACT

The paper discusses work in developing a prototype of an intelligent software environment to support system design and simulation activities. Knowledge-Based System Design and Multifacetted Modelling are foundational basis for the development. The paper briefly presents the basic tenets of the underlying methodologies. It then describes the current state of the implementation.

1. ELEMENTS OF KNOWLEDGE-BASED DESIGN AND SIMULATION

Despite great strides in development of computational tools such as high performance workstations intended to help to cope with the rising complexity of designs, the design process remains error prone. Given the often severe constraints imposed by cost, environmental impacts, safety regulations, etc., it is a fact of life that designers are forced to make compromises that would not be necessary in an ideal world. Simulation is increasingly recognized as a useful tool in assessing the quality of sub-optimal design choices and arriving at acceptable trade-offs.

Our research aims to develop and implement a methodology of design in which design models can be synthesized and tested within a number of objectives, requirements, and constraints. This framework, termed knowledge-based system design and simulation, is presented in detail in (Rozenblit and Zeigler, 1985, 1988; Rozenblit, 1985). Here, we summarize its basic tenets.

Design objectives (understood here in a broader context that includes requirements and constraints impinging the design process) drive three fundamental processes in the methodology: first, they facilitate the construction, retrieval, and manipulation of *design entity structures* (Rozenblit and Zeigler 1986, 1988). The design entity structure is based on a tree-like graph that encompasses the boundaries, decompositions and taxonomic relationships that have been perceived for the system being modelled. An entity signifies a conceptual part of the system which has been identified as a component in one or more decompositions. Each such decomposition is called an aspect. Thus entities and aspects are thought of as components and decompositions, respectively. In addition to decompositions, there are relations termed specializations. A specialization relation facilitates representation of variants for an entity. Called specialized entities, such variants inherit properties of an entity to which they are related by the specialization relation.

Aspects can have coupling constraints attached to them. Coupling constraints restrict the way in which components (represented by entities) identified in decompositions (represented by aspects) can be joined together.

In addition to coupling constraints, there are selection constraints in the system entity structure. Selection constraints are associated with specializations of an entity. They restrict the way in which its subentities may replace it in the process of model construction. Synthesis constraints restrict ways in entities selected from specializations may me configured to represent the structure of the system being designed (Rozenblit et. al., 1986, Rozenblit and Huang 1987). In Section 2, we shall focus on the process that employs the production rule formalism to support automatic selection of entities and synthesis of a design model structure. We call this process *constraints-driven model structure generation.*

The design objectives also serve as a basis for the specification of the *generic observation frames* and *experimental frames* (Zeigler, 1984a, Rozenblit and Zeigler, 1988). Generic frame consist of input, output, and summary generic variable types. The variable types express performance indices associated with a given modelling objective. Experimental frames are instantiated generic frames wherein variable types are associated with model components and execution run conditions are defined in experiment initialization, continuation, and termination sets (Zeigler, 1984a). They are employed to evaluate performance of design models.

To perform the evaluation a simulation environment is invoked. A software shell called DEVS-Scheme is used as the simulation engine. DEVS-Scheme (Zeigler 1986, 1987a) is a knowledge-based simulation environment for modelling and design that facilitates construction of families of models in a form easily reusable by retrieval from a model base. The environment supports construction of hierarchical discrete event models and is written in the PC-Scheme language which runs on IBM compatible microcomputers and on the Texas Instruments Explorer. Model specification and retrieval in the DEVS-Scheme simulation environment is mediated by a knowledge representation component designed using the system entity structuring concepts. A user prunes the entity structure obtaining a reduced structure that specifies a hierarchical composition tree. Upon invoking the transform procedure, the system searches the model base for model components specified in the model composition tree and synthesizes the desired model by coupling them together in a hierarchical manner. The result is a discrete event simulation model expressed in DEVS-Scheme which is ready to be executed to perform simulation studies.

The basic organization of software supporting our framework is depicted in Figure 1. In the ensuing sections, we provide details concerning both the model structure generation and DEVS-Scheme simulation engine.

2. MODEL STRUCTURE GENERATION

In this section we focus on the process that employs the production rule formalism to support automatic selection of entities from taxonomic relationships and synthesis of structures underlying the simulation models.

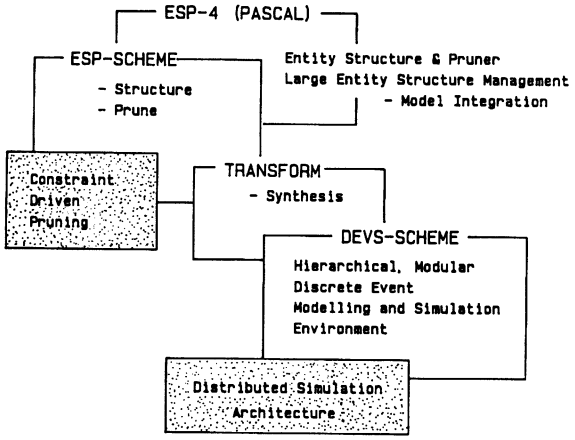

Figure 1 Organization of Software under Development.

The process consists in specifying the system entity structure for a given modelling problem. Then, a knowledge base that contains rules for selection and configuration of the entities is constructed. The modeller invokes the inference engine which, through a series of queries based on the constraint rules, allows him/her to consult on an appropriate structure for the modelling problem at hand. The result is a recommendation for a model composition tree (Zeigler, 1984a). The composition tree is used by DEVS-Scheme environment to retrieve models from the model base. The retrieved models are automatically linked in a hierarchical manner according to the coupling constraints. Figure 2 illustrates the model structure generation process in the rule-based shell called MODSYN (Model Synthesizer).

We now proceed to briefly describe MODSYN. The basic system's components are the knowledge base and the inferencing shell.

Knowledge Base Construction

The process of knowledge base construction begins with setting up the system entity structure for the model being constructed. At the present time we use previously developed tools for entity structuring (ESP4 - Entity Structuring Program (Zeigler et. al., 1980)). The system entity structure is a basis for what we term a *conceptual network*. This is a declarative representation of modelling domain objects.

The production rule formalism is used to express modelling objectives, constraints, and requirements. Domain experts provide knowledge about admissible choices of design components and their combinations, design data regarding expected performance given a particular component choice, etc. A detailed example of a rule base for a local area network design problem is given in (Rozenblit and Huang 1987).

To prune the system entity structure, we generate the following rule sets:

Selection rule set: each selection rule stands for a choice of an entity in a specialization.

Synthesis rule set: after selection rules have been applied to the entity structure, synthesis rules ensure proper configuration of the selected entities. They also coordinate the actions of the selection rules. Certainty factors are are employed to indicate the applicability of the rules.

Selection rules are associated with the entities whereas the synthesis rules are attached to the aspects of the domain entity structure. Each rule set can be regarded as a module. Therefore the entire rule base is constructed in a *hierarchical manner* imposed by the entity structure. We believe such a hierarchical structure is necessary to increase the efficiency of pruning in systems with a large number of rules.

To reduce the number of links between modules in the hierarchically organized rule base, we allow for multiple actions (conclusions) in the rule syntax. To reduce the number of modules, we connect the premises with the logical "or" or "and". The template rule syntax has the following form:

```
if   object_attribute_1 = value_1 and/or
     object_attribute_2 = value_2 and/or
                .....
                .....
     object_attribute_n = value_n

then conclusion_1 = value_1 (cf1) and
     conclusion_2 = value_2 (cf2) and
                .....
                .....
```

where cf1,cf2,..., are certainty factors whose values range from 0 which stands for no recommendation, to 1 which denotes a strong recommendation.

Inference Engine Design

MODSYN shell has been implemented in Turbo Prolog and runs on IBM PC compatible machines. The inference engine uses the strategy of "generate and test", i.e., it takes the initial data from the user and the hypothesis generated by the knowledge base to prune the search space tree. In other words, the engine attempts to match the data with the information contained in the knowledge base. If the data match, the engine climbs up the tree, trying to prove the next hypothesis. We use aspect ordering in order to eliminate aspects not desirable in the model we are constructing, and specialization-oriented pruning to select unique entities for the model composition trees. For a complete description of the shell we refer the reader to Huang (1987).

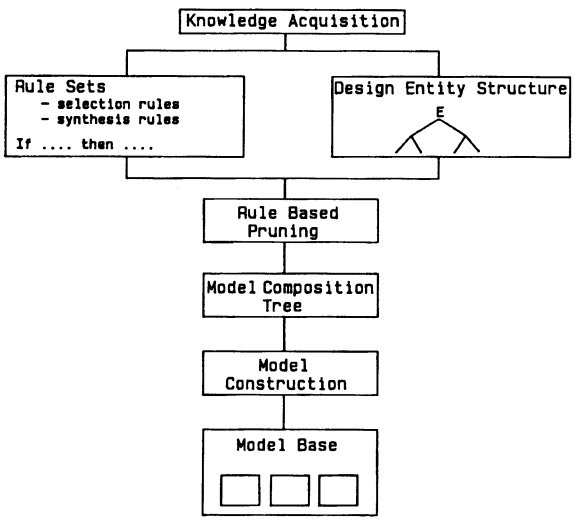

Figure 2 Model Structure Generation in MODSYS.

We have completed testing the shell and are currently porting it to a Scheme environment. This will provide a front end model processing capabilities for simulation in DEVS-Scheme.

3. HIERARCHICAL MODEL CONSTRUCTION IN DEVS-SCHEME ENVIRONMENT

DEVS-Scheme Environment

The Discrete Event System Specification (DEVS) formalism introduced by Zeigler (1976) provides a means of specifying a mathematical object called a system. Basically, a system has a time base, inputs, states, and outputs, and functions for determining next states and outputs given current states and inputs (Zeigler, 1984b).

The DEVS formalism underlies DEVS-Scheme, a general purpose environment for constructing hierarchical discrete event models (Zeigler, 1987a). DEVS-Scheme is written in the PC-Scheme language which runs on DOS compatible microcomputers and under a Scheme interpreter for the Texas Instruments Explorer. DEVS-Scheme is implemented as a shell that sits upon PC-Scheme in such a way that all of the underlying Lisp-based and objected oriented programming language features are available to the user. The result is a powerful basis for combining AI and simulation techniques.

The architecture of the DEVS-Scheme simulation system is derived from the abstract simulator concepts (Zeigler, 1984a) associated with the hierarchical, modular DEVS formalism. Since such a scheme is naturally implemented by multiprocessor architectures, models developed in DEVS-Scheme are readily transportable to distributed simulation systems designed according to such principles. Finally, since structure descriptions in DEVS-Scheme are accessible to run-time modification, the environment provides a convenient basis for development of learning or evolutionary models which adapt or change their own internal structure.

DEVS-Scheme is principaly coded in SCOOPS, the object-oriented superset of PC-Scheme. All classes in DEVS-Scheme are subclasses of the universal class entities which provides tools for manipulating objects in these classes (these objects are hereafter called entities). The inheritance mechanism ensures that such general facilities need only be defined once and for all. Entities of a desired class may be constructed using a method *mk-ent* and destroyed using a method *destroy*. More specifically, mk-ent makes the entity and places it in the list of members of the given class, *lst*; destroy removes the entity from this list. Every entity has a name, assigned to it upon creation.

Models and *processors*, the main subclasses of entities, provide the basic constructs needed for modelling and simulation. *Models* is further specialized into the major classes *atomic-models* and *coupled-models*, which in turn are specialized into more specific cases, a process which may be continued indefinitely as the user builds up a specific model base. Class *processors*, on the other hand, has three specializations: *simulators*, *co-ordinators*, and *root-co-ordinators*, which serve to handle all the simulation needs. Detail description of the class hierarchy in DEVS-Scheme is available in (Kim, 1988).

Hierarchical Model Construction

The DEVS-Scheme environment provides layer of objects and methods which may be used to achieve more powerful features. In paticular, a second layer, ESP-Scheme, implements *system entity structure* to synthesize and organize a family of models called the model base. Complete description of ESP-Scheme is beyond the scope of this paper. Details are available in (Zeigler, 1987b; Kim, 1988; Kim et. al. 1988).

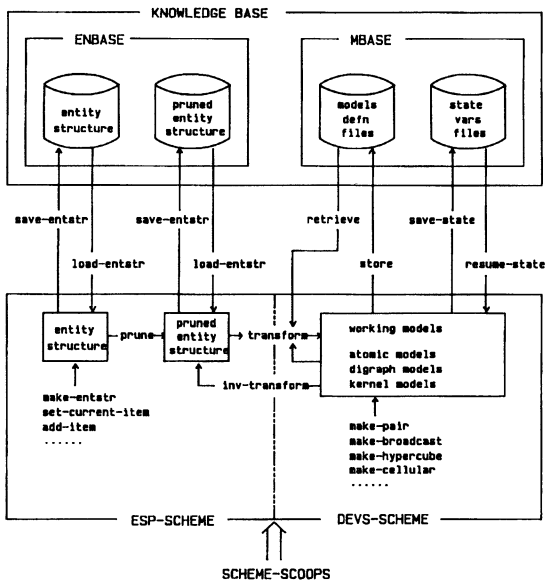

Figure 3 DEVS-Scheme Modeling /Simulation Environment.

The knowledge base framework shown in Figure 3 is intended to be generative in nature, i.e., it should be a compact representation scheme which can be unfolded to generate the family of all possible models synthesizable from components in the model base. The user, whether human or artificial, should be a goal-directed agent which can interrogate the knowledge base and synthesize a model using pruning operations that ultimately reduce the structure to a composition tree.

As shown in Figure 3, model objects expressed in DEVS-Scheme must reside in working memory in order to be simulated. Such an object can be reconstructed from disk file definitions by direct evaluation (the only possibility for *atomic-models*) or by applying the transform function to a pruned entity structure in working memory. The pruned entity structure is in turn obtained by pruning an entity structure, selecting one possibility from whole family spanned by the structure.

As it traverses the pruned entity structure, transform calls upon a retrieval process to search for a model of the current entity. If one is found, it is used and transformation of the entity subtree is aborted. Retrieve looks for a model first in working memory, then in model definition files, and finally, provided that the entity is a leaf, in pruned-entity structure files. The latter mode requires an invokation of transform which is executed in a separate Scheme environment so as not to interfere with the parent environment (see Kim, 1988 for greater detail).

Manipulation of Complex Hierarchical Structures

Since models in DEVS-Scheme may be complex, hierarchical structures special attention has been paid to replicating such structures. To test the methods for creating copies of models we employ a novel approach: we implemented a parallel set of methods for checking isomorphism between models. The criteria for correct copying are formalized in the isormphism methods. For a copying method to be valid, a copy of a model must be isomorphic to the original as determined by the isomorphism test.

As we have seen above, isomorphic copies of existing models are needed to conveniently construct complex coupled models. DEVS-Scheme provides two main alternatives for creating such copies. The first method, *make-new*, when sent to a model creates an isomorphic copy of the original which is an instance of the same class as the original. The primary classes (*atomic-models*, *digraph-models*, and the specializations of *kernel-models*) require their own versions of the make-new method since each has features that are unique to itself. (Sub-classes of these primary classes, if they do not add additional structure, can inherit the make-new method from the primary class.) Since *coupled-models* instances are hierarchical in structure, the make-new method must be recursive in the sense that components at each level must replicate themselves with their own make-new methods.

The second method, *make-class*, when sent to a model, creates a class definition with the original model as template. Instances created in such a class will be isomorphic to the original. However, in contrast to the effect of make-new, such instances are members of a different class than the original. For example, for an atomic-model m, consider the following:

```
(send m make-class 'ms)
(mk-ent ms 'n).
```

The first command will create a class named ms whose instances are isomorphic to m. The second will create an instance of ms called n. Note however, that m is an member of *atomic-models* while n is a member of class ms. Method make-new may be employed whenever an isomorphic copy of a model is desired. Method make-class must be employed in order to establish a class to serve as the kernel class for an instance of *kernel-models*. For example, to create an instance of *broadcast-models* to contain components all isomorphic to an existing model m, we require a class with m as template. Note that we can create different instances of such *kernel-models* each having a different class, but all classes having m as template. For example, two networks of IBM PCs may be modeled as distinct instances of *broadcast-models*, as in:

```
(send m ibm-pc make-class 'ibm-pc1s)
(send m ibm-pc make-class 'ibm-pc2s)
(make-broadcast ibm-pc1s)
(make-broadcast ibm-pc2s).
```

The last two commands create the distinct broadcast models, br-IBM-PC1S and br-IBM-PC1S respectively. In an example application, these two broadcast models may be linked together as components in a digraph-model to represent gate-way connected local area networks.

In general, there may be any number of instances of *kernel-models* having "isomorphic" classes, i.e., classes whose instances are all isomorphic to each other. Operation of the method make-members of *kernel-models* (referred to above) can now be explained. Consider the following:

```
(make-hypercube ms)
(send hc-MS make-members 'c 3).
```

The first command makes a hypercube-model hc-MS with kernel class ms and *init-cell* an instance of ms (using *mk-ent*). The second command causes the sequence:

```
(send init-cell make-new 'c0)
(send init-cell make-new 'c1)
(send init-cell make-new 'c2)
```

which creates objects c0, c1, and c2 each isomorphic to init-cell (hence to each other) and belonging to the same class as init-cell, namely the kernel class ms. Detail algorithms for testing model isomorphism are available in (kim, 1988).

4. CONCLUSIONS

This paper further extends our research into the methodology of model development and simulation. We have augmented system entity structure pruning algorithms with a rule-based process for selecting and synthesizing model objects representing model components. This process is driven by the modelling project's requirements and constraints. Therefore, we are now able to assist the modeller in choosing and properly configuring the model components.

Implementation of the DEVS hierarchical, modular formalism in DEVS-Scheme has opened up a wealth of possibilities for investigating methodology-based support of modelling and simulation. The symbol manipulation and object-oriented facilities of Scheme make it relatively easy to code complex structures and operations on them. Since Scheme (as is its parent, LISP) is a "language to develop languages in," an environment can be evolved in which tools are readily developed and integrated. As the range of toos discussed here indicates, we have found Scheme to be an excellent medium for tool development. In contrast a compiled language can not as easily support such environment evolution.

Another development in integrating the environment is currently under way. Rozenblit and Hu (1988) are developing procedures for automatic experimental frame generation from a repository of basic frame components, called frame base. Such procedures will be employed during performance evaluation of design models.

REFERENCES

Huang, Y.M., (1987) "Building an Expert System Shell for Model Synthesis in Logic Programming," M.S. thesis, Dept. of Electrical and Computer Engineering, The University of Arizona, Tucson, AZ

Kim, Tag Gon (1988), "A Knowledge-Based Environment for Hierarchical Modelling and Simulation", Doctoral Dissertation, University of Arizona, Tucson.

Kim, Tag Gon, Guoqing Zhang, Bernard P. Zeigler (1988), "Entity Structure Management of Continuous Simulation Models", in *Proc. Summer Sim. Conf.*, Seattle, 1988.

Nilsson, N.J., (1980) *Principles of Artificial Intellignece*,Tioga, Palo Alto, CA.

Rozenblit, J.W., Zeigler, B.P., (1985) "Concepts for Knowledge-Based System Design Environments," *Proc. of the 1985 Winter Simulation Conference*, San Francisco, CA.

Rozenblit, J.W., Zeigler, B.P. (1987) "Design and Modelling Concepts," in *Encyclopedia of Robotics*, John Wiley, N.Y.

Rozenblit, J.W., Zeigler, B.P., (1986) "Entity-Based Structures for Model and Experimental Frame Construction," in *Modelling and Simulation in Artificial Intelligence Era* (ed. M.S. Elzas et. al.) North Holland, Amsterdam.

Rozenblit, J.W. (1986) "A Conceptual Basis for Integrated, Model-Based System Design," Technical Report, Dept. of Electrical and Computer Engineering, University of Arizona, Tucson, January 1986.

Winston, P.H., (1984) *Artificial Intelligence*, Addison-Wesley, Reading, MA.

Zeigler, B.P., Belogus D., Bolshoi, A., (1980) "ESP - An Interactive Tool for System Structuring," *Proc. of the 1980 European Meeting on Cybernetics and Systems Research*, Hemisphere Press.

Zeigler, B.P. (1976), *Theory of Modelling and Simulation*, Wiley, NY. (Reissued by Krieger Pub. Co., Malabar, FL. 1985).

Zeigler, B.P. (1984a) *Multifacetted Modelling and Discrete Event Simulation,* Academic Press, London.

Zeigler, B.P.(1984b) "System-Theoretic Representation of Simulation Models," *IIE Transactions*, March, pp.19-34.

Zeigler, B.P. (1987a), "Hierarchical, Modular Discrete Event Modelling in an Object Oriented Environment," *Simulation J.* Vol. 49:5, pp. 219-230.

Zeigler, B. P. (1987b), "Knowledge Representation from Minsky to Newton and Beyond," *Applied Artificial Intelligence*, vol.1 87-107, Hemisphere Pub. Co.

Jerzy W. Rozenblit is an assistant professor in the Electrical and Computer Engineering Department at The University of Arizona. He received his Ph.D. in Computer Science from Wayne State University in Detroit, in 1985. His research interests are in the areas of modelling and simulation, system design, and artificial intelligence. He is a member of ACM, IEEE Computer Society, and The Society for Computer Simulation.

Jerzy W. Rozenblit
Dept. of Electr. and Computer Engr.
The University of Arizona
Tucson, Arizona 85721
(602) 621-6177

Tag Gon Kim is a research engineer at the Environmental Research Lab of the University of Arizona. From 1980 to 1983, he has been a faculty in the Department of Electronics and Communication Engineering at the National Fisheries University of Pusan, Korea. His research interests are in the areas of AI, modelling and simulation, computer architectures, and expert system based real-time control system design. He received his Ph. D. in Computer Engineering from the University of Arizona. He is a member of IEEE, ACM, and SCS.

Tag Gon Kim
ERLab, The University of Arizona
2601 E., Airport Dr.
Tucson, AZ 85706
(602) 741-1990

Bernard P. Zeigler is a professor of Computer Engineer at the University of Arizona. He is the author of *Multifacetted Modelling and Discrete Event Simulation*, Academic Press, 1984, and *Theory of Modelling and Simulation*, John Wiley, 1976. His research interests include artificial intelligence, distributed simulation, and expert system for simulation methodology.

Bernard P. Zeigler
Dept. of Electr. and Computer Engr.
The University of Arizona
Tucson, AZ 85721
(602) 621-2108