Proceedings of the 1989 Winter Simulation Conference
E.A. MacNair, K.J. Musselman, P. Heidelberger (eds.)

Hierarchical, Modular Modelling in DEVS-Scheme

Bernard P. Zeigler, Jhyfang Hu, and Jerzy W. Rozenblit
Dept. of Electrical and Computer Engineering
University of Arizona
Tucson, AZ 85721

ABSTRACT

This tutorial describes the features of DEVS-Scheme a knowledge based simulation environment for hierarchical, modular discrete event models. The attributes of DEVS-Scheme which distinguish it from conventional approaches are described in terms of a set of layers, each layer dependent on its predecessor to achieve its functionality.

INTRODUCTION

Hierarchical, modular specification of discrete event models offers a basis for reusable model bases and hence for enhanced simulation of truly varied design alternatives. This tutorial summarizes the DEVS-Scheme environment, which implements the DEVS formalism for hierarchical, modular models. DEVS-Scheme is implemented in PC-Scheme, a powerful LISP dialect for microcomputers containing an object-oriented programming subsystem. Since both the implementation and the underlying language are accessible to the user, the result is a capable medium for combining simulation modeling and artificial intelligence techniques. The environment is developed in an object-oriented manner which lends itself to model base organization using the entity structure knowledge representation. It also serves as a medium for developing hierarchical distributed simulation models and architectures.

The features of DEVS-Scheme can be better understood by organizing them within a set of layers that characterizes its software design structure. In Figure 1, Scheme and SCOOPS, the Lisp-based, object-oriented programming system provides the foundation on which the system is built. The properties of this lowest layer make it possible to realize similar properties at the higher layers. For example, the ability to test the behavior of an object in stand-alone fashion is responsible for the same testability property that obtains for model objects on Layer 1. The next layer, which supports systems model construction, acquires many of its properties from both Layer 0 and the systems concepts embodied in the DEVS formalism. Layer 2 relies on Layer 1 to provide the ability to specify models which populate the model base that Layer-2 organizes. The highest layer, that of systems design, augments the system entity structure knowledge representation with knowledge related to model-based design.

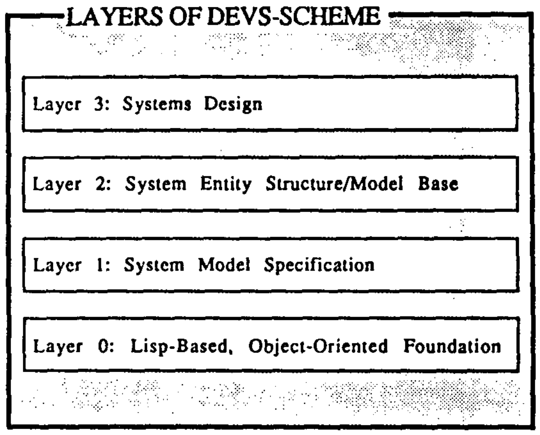

LAYERS OF DEVS-SCHEME
Layer 3: Systems Design
Layer 2: System Entity Structure/Model Base
Layer 1: System Model Specification
Layer 0: Lisp-Based, Object-Oriented Foundation

Figure1 Layers of DEVS-Scheme

Layer-0: Lisp-based, Object-Oriented Foundation

Major features of the Layer-0 DEVS-Scheme are:

Symbolic and Numeric Processing: Number crunching ability is typically a requirement of traditional models but symbol manipulation capability is essential to knowledge based simulation characterized by intelligent components and/or intelligent model manipulation. In contrast to many artificial intelligence studies, which stay at a high degree of abstraction, knowledge-based simulation, e.g., of robotic systems, must have efficient substrates of both kinds of processing support.

Software Design Attributes: The object concept embodies the fruits of software engineering research such as encapsulation (packaging of data and code together), abstraction (the objects behavior as seen through its methods), and information hiding (the structural detail it hides).

Environment Development Support: The symbol manipulation and object-oriented facilities of Scheme make it relatively easy to code complex structures and operations on them. Since Scheme is an interpretted language, it combines levels that would be separated in the translation and execution steps of compiled languages. Thus, like its parent, LISP, Scheme is a "language to develop languages in". In it, an environment can be evolved in which tools are readily developed and integrated. In contrast a compiled language can not as easily support such environment evolution since one must work at both the language and operating system levels to do this.

Testability: An object can be tested against its behavioral requirements specification (e.g., given in axiomatic form) by injecting sequences of messages and comparing the object's response with that expected.

Extensibility: A software system can evolve incrementally by addition of new classes without disturbing previously written code.

Replicatability: Since object structures are well defined by their class templates, class specific methods can be written so that objects, no matter how complex, can be easily replicated (copied).

Concurrent Implementation: It is quite natural to extend object behavior so that all objects are simultaneously active; the parallelism in such concurrent object-oriented systems can then be exploited by mapping to multiprocessor architectures.

Browsability: The ability allows a user to browse easily through the class hierarchy of a system, getting both global and local perspectives. Unfortunately, in contrast to SMALLTALK and LOOPS, this is not directly available in SCOOPS.

Layer-1: Systems Model Specification

Major features of the Layer-1 DEVS-Scheme are:

System Theoretic Formal Basis: The DEVS formalism for discrete event systems is directly implemented as the means of expression in DEVS-Scheme. This provides a sound semantics for discrete event model representation and basis for mathematical and other symbolic processing of model specifications.

Model Specification Language: As a set theoretic construct, the DEVS formalism by itself is not a practical means of specifying models. However, DEVS-Scheme supports the structure of the DEVS-formalism with the underlying expressive power of Scheme, thus offering a combination of the best of both worlds: formal specification with ease of model development.

Modularity: Model specifications are self-contained and have input and output ports through which all interaction with the external world must take place. Models, as objects, have the software engineering attributes inherited from Layer-0. In addition, ports provide a level of delayed binding which needs to be resolved only when models are coupled together.

Closure under Coupling: Models may be connected together by coupling of input and output ports to create larger, coupled models, having the same interface properties as the components. Hierarchical construction follows as a consequence of modularity and closure under coupling; successively more complex models can be built by using, as building blocks, the coupled models already constructed.

Stand-alone and Bottom-up Testability: Due to object encapsulation and input/output modularity, models are independently verifiable at every stage of hierarchical construction. This fosters secure and incremental bottom-up synthesis of complex models.

Experimental Frame/Model Separation: Experimental frames are independently realized as models of special kinds: generators, transducers, acceptors. Having input/output ports, they can be coupled to models to which they are applicable.

Isomorphic Replicatability: Copies can easily be made of complex, hierarchical models, with consistent name assignments, as components in homogeneous models. Systems isomorphism concepts provide the formal basis for correctness of model replication.

Hierarchical Distributed Simulation: As objects, models can be executed on concurrent object-based processors; however, using the abstract simulator concepts, more advantageous hierarchical multiprocessor systems can be designed and hierarchical models mapped to them so that maximum speed up is obtained. DEVS-Scheme provides timing measurements from its underlying "virtual multiprocessor" simulation engine to support analysis for optimal multiprocessor mappings.

System Manipulations: Derived from the formal structure and system theoretic basis is the ability to implement systems operations such as structure transformations, tests for homomorphism, etc.

Model Abstraction and Simplification: Systems model description, as rendered by the DEVS formalism, facilitates tool development for DEVS representation of continuous systems which can be used for faster simulation and for event-based control. Simplification of DEVS multicomponent models (coupled models) can be obtained by conversion to equivalent atomic-models, from which a homomorphic lumped model is constructed. The motivation is to obtain faster running lumped models that can replace a component of a hierarchical model so that its complement can be more efficiently run and tested.

System Specification Formalisms within DEVS-

Scheme: The hierarchical, modular modelling and simulation concepts first developed in the DEVS formalism can be implemented in continuous and discrete-time formalisms and combined with DEVS to obtain multiformalism simulation.

Model specification extensibility: New kinds of model specification formats can be readily added as specialized classes of atomic and coupled models

Rule-based Modelling: The specialized class, forward-models provides the ability to specify models as sets of activities, which are have a rule-like character and combine symbolic and dynamic model specification.

Granularity: Activities within forward-models provide the most granual level of specification or knowledge representation in DEVS-Scheme. Sets of activities can be inherited and combined to form larger sets.

Layer-2: System Entity Structure/Model-Base

Major features supported by the Layer-2 of DEVS-Scheme include:

Axiomatic Specification: The system entity structure is a treelike graph which is formally characterized in an axiomatic manner thus facilitating design and verification of the complex operations that are required to support model generation and reuse.

Synthesis Constraints: The system entity structure can be augmented with rules that enforce constraints on selections from specializations. Such constraints can be of a global character where selections in various parts of the SES must be correlated. They also can be sensitive to the context in which the selection is being made. This supports coherence of the entity structure base.

Model Synthesis via Pruning: Hierarchical models can synthesized by creating and transforming pruned entity structures. This requires only that the lowest level atomic-models referenced by a pruned entity structure reside in the model base.

Archivability: Models can be saved on disk in the form of model definition files or as pruned entity structure files. The latter form is preferred both for the convenience of model construction afforded as well as for taking advantage of the automatic cataloguing provided by the environment.

Model-base Cataloguing: Pruned entity structures are given the name of their root entity together with a user supplied suffix so that they can be identified as representing alternative models of the root entity.

Reusability: Models developed for studying a particular real system, archived in the model base and managed by the system entity structure, are retrievable for use as components in new models. Due to pruned entity naming, hierarchical models expressed in the form of pruned entity structures are just are reusable as those in the model base. This fosters accumulative growth of "off-the-shelf" reusable components.

Layer-3: Systems Design

The systems design layer supports model-based design. It is implemented by embedding the system entity structure in a richer frame-based knowledge representation scheme called Frames and Rules Associated System Entity Structure (FRASES). A great amount of information can be attached to nodes in the SES which can be used to drive the system design process. The following briefly outlines features of system design in the DEVS-Scheme environment.

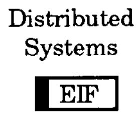
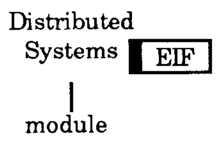
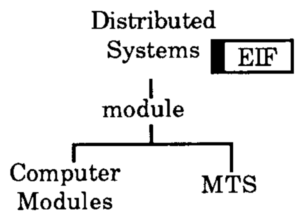
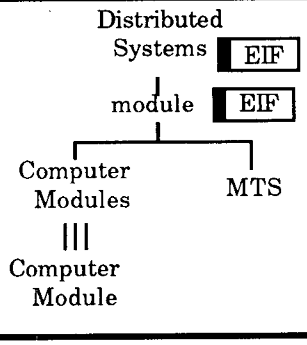
Knowledge Acquisition by Representation (KAR): The FRASES organization constitutes a set of hypotheses on what it is important for systems design. To the extent that these hypotheses are correct and complete, capturing this knowledge can be facilitated using FRASES as a template for conducting a dialog with a system design expert. A sample dialog is illustrated in Figure 2.

Goal Driven Pruning: Layer-3 helps users with design model construction by providing guidance in pruning sensitive to their design goals. Rules are attached to FRASES to represent selection and synthesis knowledge. Selection rules attached to specializations specify entity choices in terms of local design criteria. Synthesis rules attached to decompositions constrain the plausable combinations of model components in decompositions. Different inference engines characterizing forward, backward, and weight-oriented chaining, were developed to facilitate the design reasoning process.

Hierarchical Design Specification: Design requirements including constraints, objectives, and criteria preference can be specified and attached to each corresponding entity at different design abstraction levels. This enables a hierarchical design specification.

Automatic Performance Modeling: To help develop math-intensive models for measuring performance of design models, Layer-3 of DEVS-Scheme supports automatic generation of experimental frames. By providing the algebraic expression of a defined performance index, atomic frames are extracted automatically and aggregated into an experimental frame model which can be coupled to the design model for performance evaluation.

Automated Design Evaluation and Ranking: Design models synthesized via pruning are automatically subjected to simulation evaluation in performance-relevant experimental frames and ranked according to multicriteria decision methods. FRASES attaches slots with

KAR/EXPERT INTERACTION	FRASES CONVERSION
KAR> What is your problem domain ? => Distributed-Systems	Distributed Systems
KAR> What kind of static attributes do you want to add to the system? => (designer Jeff) (date 4/19/89) (place UA) KAR> What kind of design parameters will be considered at this design level? => cost size KAR> What kind of performance indices will be considered to evaluate a Distributed-System • • •	
KAR> Can you classify a Distributed-System based on certain specialization? =>why KAR>->> This question is used to query how experts ->> classify the variants of the entity asked for. For ->> example, a computer network can be clssified ->> into RING, BUS, and TREE based on Topology. KAR> Can you classify a Distributed-System based on certain specializations? => nil KAR> Can you decompose a Distributed-System based on certain aspect? => module	
KAR> What are these subcomponents when you decompose a Distributed-System based on Module? => Computer-Modules Message-Transfer-System (MTS)	 Distributed Systems EIF module Computer Modules MTS
KAR> Does the number of Computing-Modules vary with design requirements? => yes KAR> Specify the range for the number of Computing-Modules? => 0 64 *;;; multiple decomposition* • • •	 Distributed Systems EIF module EIF Computer Modules MTS Computer Module

- continued -

KAR> Can you rank the design priority for Processor, Memory, and I/O-Devices?
=> Processor Memory I/O-Devices

• • •

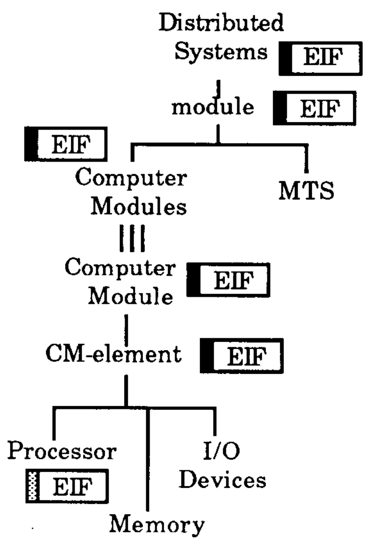
KAR> What kind of performance indices will be employed to evaluate a Computing-Module?
=> response-time thruput

• • •

KAR> Can you classify a processor based on certain specialization?
=> nil

KAR> Can you decompose a processor based on certain aspect?
=> nil
;;; satisfied with the level of design abstraction
;;; for processor

• • •



KAR> Can you classify a Message-Transfer-System based on certain specializations?
=> technology

KAR> What are the alternatives when you classify a Message-Transfer-System based on technology?
=> IN LAN

KAR> Can you specify selection rules that determine MTS technology?
=>r1: if interaction between computing modules is high or resources sharing capability is required then select LAN.
r2: if interaction between computing modules is low or resources sharing capability is not required then select IN.

• • •

KAR> Will the Distributed-Systems use a LAN or IN as the Message-Transfer-System (MTS) and consist of a variable number of computer modules?
=> yes *;;; verifying inheritance*

• • •

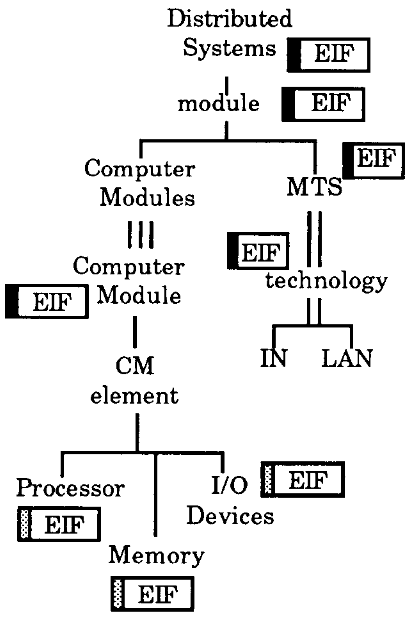


Figure 2 Illustration of KAR with FRASES

nodes for specifications of performance indexes, the system measurements they are based on, the experimental frames that can acquire data for such measurements, and the trade-off criteria for ranking of alternatives.

CONCLUSION

DEVS-Scheme was developed at the University of Arizona as a product of research funded by the National Science Foundation. It is currently being used to design autonomous robots for laboratory automation in a project sponsored by the AI Research Center, NASA Ames. The environment is available for academic and industrial use and is being commercialized by SIMEX, a corporation offering simulation methodology for knowledge based control.

REFERENCES

Hu, Jhyfang (1989), "Towards A Knowledge-Based Design Support Environment For Design Automation and Performance Evaluation", Ph.D. Dissertation, University of Arizona, Tucson, Arizona.

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

Rozenblit, J. W. (1985), "A Conceptual Basis for Model-Based System Design", Ph.D. Dissertation, Wayne State University, Detroit, MI.

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

Zeigler, B. P. (1984), "Multifacetted Modelling and Discrete Event Simulation", Academic Press, London and Orlando, FL.

Zeigler, B. P. (1989), "Hierarchical Modular DEVS: Model Knowledge and Endomorphy in Object-Oriented Simulation", Academic Press, Boston.

Authors' Biographies

Bernard P. Zeigler is a professor of Electrical and Computer Engineering at the University of Arizona. He is author of "Hierarchical Modular DEVS: Model Knowledge and Endomorphy in Object-Oriented Simulation", Academic Press, 1989, "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.

Jhyfang Hu received his M.S. and Ph.D. degrees in Electrical and Computer Engineering from University of Arizona in 1986 and 1989. His primary research interests include AI/CAD VLSI design and testing, object-oriented database management, and expert systems for design automation and performance evaluation. He is a member of Eta Kappa Nu and IEEE Computer Society.

Jerzy W. Rozenblit is an assistant professor of Electrical and Computer Engineering at University of Arizona, Tucson. He received his Ph.D. degree in Computer Science from Wayne State University in Detroit in 1986. His research interests are in the area of modelling and simulation, system design, and artificial intelligence. He is a member of ACM, IEEE Computer Society, and Society of Computer Simulation.