System Entity Structuring and Model Base Management


Abstract—System entity structure (SES) is a structural knowledge representation scheme that contains knowledge of decomposition, taxonomy, and coupling of a system. Formally, the SES is a labeled tree with attached variable types that satisfy certain axioms. Described is a realization of the SES in Scheme (a Lisp dialect) called ESP-Scheme. The computer representation of SES and main operations on SES are presented, and then facilities provided by ESP-Scheme are described. Two examples of application are discussed: a parallel processor model and a simulation study of a university phone registration system. ESP-Scheme acts as a model base management system in DEVS-Scheme, a knowledge-based simulation environment. It supports specification of the structure of a family of models, pruning the structure to a reduced version, and transforming the latter to a simulation model by synthesizing component models in the model base.

I. INTRODUCTION

THE NEED for greatly enhanced support of simulation modeling activities is increasingly being recognized. Model development methodology, state-of-the-art software technology, and artificial intelligence, especially knowledge-based concepts [6], [8], [14]–[16], [18], [23], [26] have been suggested as means to achieve advanced simulation environments [1], [2], [10], [17].

To cite one application context, with the new importance given to simulation in military training and analysis, there is strong demand for a simulation workbench, i.e., "an evolving architecture providing an integrated collection of standards, protocols, and tools to enable analysts...to build, understand, use and reuse simulations..." [9]. Such a workbench would have to provide a sharable repository of models and a means of assisting users to browse through, and understand the content of, stored models to confidently select those relevant to the current analytic task. Although this need for model base management is recognized, there are few concrete proposals to meet it [9].

The model base management system discussed in this paper, based on the System Entity Structure, is the only one, to the authors' knowledge, that provides a working software system capable of addressing the needs of model repositories just stated. We note that "model management" here refers to the storage, cataloging, and reuse of component models to synthesize simulation models to meet current objectives. A second sense of the term, related though distinct, is used in decision support systems, where "models" refer to more generalized mathematical and analytic tools for operating on input data to produce output helpful in decision making [25].

Our approach to model base management stems from the need to develop computerized support for multifaceted modeling methodology [33]. This methodology recognizes that most systems of interest require a multiplicity of models to be developed since a single all-encompassing model, however desirable as a conceptual goal, is not a practical object. By decomposing questions and objectives into an ordered structure of experimental frames (specifications of experimentation conditions), useful partial models may be constructed, validated, and employed, each attuned to a limited set of frames.

A multifaceted model base may contain a variety of models and associated formalisms [34] that are organized with the help of the system entity structure (SES), which directs the synthesis of models from components in the model base. The SES is a knowledge representation scheme that combines decomposition, taxonomic, and coupling relationships. Knowledge representation is generally accepted to be a key ingredient in designing artificial intelligence software. Previous work identified the need for representing the structure and behavior of systems in a declarative scheme related to frame-theoretic and object-based formalisms [32], [8], [27]. The elements represented are motivated, on the one hand, by systems theory [28], [29] concepts of decomposition (i.e., how a system is hierarchically broken down into components) and coupling (i.e., 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 [8], [27]. 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 models for a given domain.
One application of this framework is to the design of systems. Here the SES serves as a means of organizing the possible configurations of a system to be designed which may be extracted with a pruning process [11], [13], [33]. Pruning is a goal-directed process, where the goal is formulated by the modeler (designer) to meet the system design requirements. Pruning reduces the set of candidate models to those suitable for the problem under study.

The implementation described here adheres to the formal characterization of the SES given in [35]. One main difference with earlier implementations [3], [7] is that the present implementation is fully integrated with the DEVS-Scheme environment that provides the underlying model synthesis and simulation layer. In addition, several powerful features to assist in combining structures to form larger ones have been added. Although space precludes a full exposition of SES concepts, the reader will find a brief review followed later by examples that illustrate the major features.

DEVS-Scheme realizes Zeigler’s DEVS (discrete event system specification) formalism in Scheme (a Lisp dialect) [12], [13], [31]. The environment supports building models in a hierarchical, modular manner, a systems-oriented approach not possible in conventional languages [5]. To organize the complex hierarchical structures of models developed using DEVS-Scheme, a model base management system is highly desirable. The system entity structuring formalism developed by Zeigler [33] is one such tool for the model base management. ESP-Scheme is a realization of the system entity structuring formalism in the Scheme environment. ESP-Scheme supports hierarchical specification of the structure of a family of models, pruning the structure to a reduced version, and transforming the latter to a simulation model by synthesizing component models in the model base developed using DEVS-Scheme.

This paper first reviews the system entity structuring formalism and then describes some overall features of the ESP-Scheme, including representation and operations of the system entity structure. It also presents an outline of a knowledge-base framework for modeling and simulation, based on the entity structure and model base. The paper illustrates the concepts presented with two examples: a parallel processor model and a simulation study of a university phone registration system.

II. THE DEVS FORMALISM AND DEVS-SCHEME

The DEVS (discrete event system specification) formalism developed by Zeigler supports specification of discrete event systems in hierarchical, modular fashion. In the formalism, one must specify 1) basic models from which larger ones are built, and 2) how these models are connected together in hierarchical fashion. Definitions for a basic model, called an atomic DEVS, and a general form of model, called a coupled DEVS, can be found in [33].

![Class hierarchy of DEVS-Scheme](image)

DEVS-Scheme, a general-purpose modeling and simulation environment, is an implementation of DEVS formalism in a LISP-based, object-oriented programming system [31], [13]. 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 object-oriented programming language features are available to the user. The result is a powerful basis for combining AI and simulation techniques.

The class specialization hierarchy in DEVS-Scheme is shown in Fig. 1. All classes in DEVS-Scheme are subclasses of the universal “class entities” that provide 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.

Models and processors, the main subclasses of entities, provide the basic constructs needed for modeling 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 that may be continued indefinitely as the user builds up a specific model base. The class definitions for atomic-models and coupled-models closely parallel those for atomic DEVS and coupled DEVS, respectively. Class “processors,” on the other hand, have three specializations: simulators, coordinators, and root coordinators. The simulators, coordinators, and root coordinators carry out the simulation of a model in a manner following the hierarchical abstract simulator concepts in [33]. Thus the
DEVS-Scheme environment

- supports modular hierarchical model construction,
- allows independent testing of components models,
- separates models from experimental frames,
- supports distributed simulation.

A detailed description of all classes in DEVS-Scheme is available in [34] and [13]. Examples will be provided to illustrate salient features.

III. SYSTEM ENTITY STRUCTURING FORMALISM: A REVIEW

A system entity structure (SES) is a knowledge representation scheme that contains the decomposition, coupling, and taxonomy information necessary to direct model synthesis [33], [19]. Formally, the SES is a labeled tree with attached variable types that satisfy five axioms: alternating mode, uniformity, strict hierarchy, valid brothers, and attached variables. A detailed discussion of the axioms is available in [34].

A. Three Relationships in SES

There are three types of nodes in the SES—entity, aspect, and specialization—which represent three types of knowledge about the structure of systems. The entity node, having several "aspects" and/or "specializations," corresponds to a model component that represents a real-world object. The aspect node (a single vertical line in the labeled tree of Fig. 2) represents one "decomposition," out of many possible, of an entity. Thus the children of an aspect node are entities, distinct components of the decomposition. The specification node (double vertical arrow in the labeled tree of Fig. 2) represents a way in which a "general" entity can be categorized into "special" entities. As shown in Fig. 2, attached to an aspect node is a coupling scheme, which specifies external input, external output, and internal couplings of a system and its components.

A "multiple entity" is a special entity that consists of a collection of homogenous components. We call such components a multiple decomposition of the multiple entity. The aspect of such a multiple entity is called multiple aspect (triple vertical lines in the labeled tree of Fig. 2). The representation of such a multiple entity is as follows. A multiple entity BS and its components B are represented by BS, three vertical lines, and B from the top down. Note that instead of presenting all B's for BS's components, only one B is placed in the labeled tree. The number of B's is specified by a variable, which is attached to the multiple aspect node.

B. Operations on SES

An SES represented by a labeled tree consists of branches and nodes (nodes are also called items). An item in the SES is one of the three types: entity, aspect, or specialization. Some of the operations performed on the SES are adding an item to the SES, deleting an item from the SES, attaching variables to items in the SES, deleting variables from items in the SES, pruning the SES, and transforming the pruned SES into a model.

The construction of a SES is a sequence of adding new items—entities, aspects or specializations—to the SES. The deletion operation, which deletes entities and associated branches from the SES, can be applied only to those entities with no aspects. The pruning operation extracts a substructure of the SES by selecting one aspect and/or one specialization for each entity in the SES. The pruning operation ultimately reduces the SES to a composition tree that contains all the information about the structure of the reduced version of the model. The transform operation synthesizes the reduced version of the model in a hierarchical fashion from components in the model base.

IV. IMPLEMENTATION OF ESP-Scheme

A. Representation of SES in ESP-Scheme

The SES is implemented by a module called "entity-structure"—a package of hidden variables and associated operations—as shown in Fig. 3. Lists of items and branches are main variables representing a tree structure for the SES. The variable "current-item" points to the current item in the SES, under which new items can be added. Each item in the items-list is represented by a structure type called item, the fields of which include "type," "name," "coupling," "multi-coup-type," and "attributes-lst." Each branch in the branches-list is represented by another structure type called branch, which maintains an ordered pair of two items, left- and right-items, in the SES.
The field "type" in item structure represents the type of an entity in an entity structure whose range is [entity, aspect, specialization]. The field "name" is used to identify an entity by its name. The field "coupling" is used to specify a coupling scheme of a model specified by a system entity structure. The coupling scheme is a collection of three coupling specifications: external input, external output, and internal coupling. Each of three coupling specifications is represented by a set of ordered pairs of ports. The representation of the coupling scheme is compatible with that of DEVS-Scheme. The field "sub-type" maintains a list of such attributes, each of which is a pair of variables and its value.

The main operations on the SES are "set-current-item," "add-item," "add-mult," "delete-item," "add-coupling," "prune," and "transform." The construction of a system BUF-PROC SES will be used to illustrate the operations. BUF-PROC is a processing element containing a buffer cascaded with a processor. The type of buffer is assumed to be either FIFO (first-in first-out) or LIFO (last-in first-out), and will be selected by the user in the pruning process. Once the SES of the BUF-PROC is built, we prune the BUF-PROC entity structure and transform the pruned BUF-PROC into the desired model. The first step in the creation of an SES is defining the root entity. Line (1) of Fig. 4(a) illustrates the creation of an entity structure named BUF-PROC. Once the SES with the root entity BUF-PROC has been created, items are added to the SES. However, a sequence of adding items should be such that the resulting SES satisfies the axiom of "alternating mode." SES axioms are automatically checked by the entity structure module as the operations are processed. Since the next items to be added are either aspects or specializations, we add an aspect called comp-dec under the root entity BUF-PROC (line (2) of Fig. 4(a)). To add other items under the aspect comp-dec requires setting the current item to the aspect comp-dec (line (3) of Fig. 4(a)). After the current item is set, two components, a buffer and a processor, are added one by one (lines (4) and (5) of Fig. 4(a)). Note that the current item of the SES is still at the aspect comp-dec. When an item with type specialization is added under the entity BUFFER, the current item must be set to the BUFFER (line (6) of Fig. 4(a)). Then line (7) adds a specialization, "buf-type," under the entity BUFFER. Similarly, lines (8), (9), and (10) add two items, FIFO-BUF and LIFO-BUF, under the specialization "buf-type."

The coupling scheme of the BUF-PROC system, which is attached to the aspect comp-dec, can be specified by the operation "add-coupling." This operation requires the specification of two entity names and their respective port names. The operation "add-coupling" specifies both internal and external coupling of the BUF-PROC system. Lines (12) and (13) of Fig. 3(a) specify the external coupling scheme, while lines (14) and (15) of Fig. 4(a) specify the internal coupling scheme of the BUF-PROC. The resulting SES for the BUF-PROC system is shown in Fig. 4(b).

A "pure" entity structure is one having no specializations and, at most, one aspect hanging from every entity. The result of pruning is a pruned entity structure, which contains fewer aspects and specializations than the original and therefore specifies a smaller family of alternative models than the latter. Ultimately, pruning terminates in a pure entity structure that specifies the synthesis of a particular hierarchical model. An example of a pruned entity structure is shown in Fig. 4(c), where FIFO-BUF, corresponding to the first-in first-out queuing discipline,
has been selected as the type of buffer desired. Note that the specialization FIFO-BUF replaces its parent, BUFFER, in all occurrences of the latter in coupling (and other) specifications.

To automatically construct a simulation model in DEVS-Scheme, we apply the “transform” operation to a pruned entity structure. Transform retrieves from the model base those models that correspond to the entities in the pruned entity structure, and then synthesizes them into a simulation model for the BUF-PROC (Fig. 4(d)). The atomic DEVs models for the FIFO-BUF, LIFO-BUF, and PROCESSOR must be stored in the model base prior to the transform operation. Details of the transform operation will be described in Section VI-C.

B. Multiple Entity

The ability to specify multiple entities and multiple decompositions provides a powerful means for representing massively parallel computer architectures which may have different connection topologies—such as broadcast, hypercube, controlled, or cellular. The specification of a parallel processor will be used to illustrate the use and power of multiple entities and multiple decompositions. Assume that the parallel processor is a collection of processing elements with one of the interconnection topologies given in the preceding, each element being a copy of the BUF-PROC that has already been specified. The parallel processor is named BUF-PROCS, which means it is a collection of BUF-PROC’s.

We can build the SES for BUF-PROCS by modifying the SES shown in Fig. 4(a). Line (1) of Fig. 4(a) is replaced by a new root entity BUF-PROCS of “multiple entity” type. A multiple aspect is added under the BUF-PROCS, and BUF-PROC is added under the multiple aspect by the operation “add-mult.” Once the three items are added, lines (21415) of Fig. 4(a) can be reused without change for the BUF-PROCS specification. The operation “add-mult-couple” specifies the internal coupling scheme for the kernel models in DEVS-Scheme in contrast to “add-coupling” for digraph models. It sets the slot “mult-coup-type” of the item of type “multiple aspect” to one of the subclasses of kernel models [11], [13], [34] in DEVS-Scheme. The resulting SES, which has a multiple entity, is shown in Fig. 5. If broadcast coupling is selected for the multiple decomposition in the pruning process, the operation “transform” will create a broadcast model of BUF-PROCS containing the number of atomic BUF-PROC models specified during the pruning process.

C. Hierarchical Model Structuring Operations

Some hierarchical model structuring operations will be discussed to show the power of the SES hierarchical model structuring formalism. Other operations can be found in [13].

The operation “add-sub-entstr” is an extension of the operation “add-item.” This operation adds an entity structure under the current item of type “aspect” in the original entity structure. Similarly, the operation “delete-sub-entstr” is an extension of the operation “delete-item.” The operation “delete-sub-entstr” deletes the subentity structure under the specified item.

The operation “add-mult” is extended by making arbitrary the number of hierarchy levels for the multiple entities to be added. The operation “add-mult-mult” allows us to specify a hierarchical construction of different kernel models. Fig. 6 shows the operation that results in three levels of hierarchy of the multiple entities AS, BS, and CS. To specify a different coupling type for different kernel models, we use the operation “set-mult-coup-type.” An application of the operation “add-mult-mult” to the modeling of a multilevel hypercube architecture can be
found in [11]. The operation “attach-num-mult-children” attaches the number of components under a multiple entity to the multiple entity.

D. Reuse of Pruned Entity Structures

The entity structure module provides several operations for reusing pruned entity structures. The operation “add-spec&ents-at-leaf” searches the entity structure base to find pruned entity structures whose root names are the same as that of a leaf entity in an entity structure. If any are found, the operation adds a specialization under the leaf entity and adds the pruned entities under the specialization (Fig. 7). The operation “mult-asp → asp” changes a multiple aspect in an entity structure to an aspect by specifying the number of children attached under the aspect.

The operation “cut-entstr” makes a nonleaf entity into a leaf entity by removing the subentity structure beneath the nonleaf entity, constructing a new entity structure and saving it in the entity structure base (we shall describe the entity structure base in Section V-A). The new entity structure has as its root the original nonleaf entity and contains the original nonleaf entity subentity structure (as shown in Fig. 8).

An entity structure for a system can be constructed in a hierarchically distributed manner so that it contains only leaf entities. Each leaf entity in the higher level entity structure may then be a root entity in a lower level entity structure. Hierarchical entity structures may be merged into a single entity structure using the operation “merge-entstr.” Merge-entstr searches for entity structures in the entity structure base with the same root names as the leaf entities of the specified entity structure. If such entity structures are present in the entity structure base, they replace the respective leaf entities in the specified entity structure.

V. FACILITIES IN ESP-SCHME

SES construction, copying, and other facilities are provided by ESP-Schme. The facility “make-entstr” creates an entity structure whose name is the same as its root name except for a prefix “e:”. For example, (make-entstr system) creates an entity structure e:system with root name system. Since the entity structure has only the root entity system, items can be added as required to construct the desired entity structure. The facility “delete-entstr” deletes an existing entity structure.

The facility “copy-entstr” copies one entity structure to another entity structure. For example, (copy-entstr e:system 'new-system) creates an entity structure named e:new-system that has the same structure (e:system). The facility copies a list of items and a list of branches from the original entity structure and constructs a new entity structure.

A list of all entity structures is maintained by an entity structure manager (described in Section VI-B). All facilities that create or delete entity structures must report to the manager the creation and/or deletion of entity structures. For example, the facility “rename-entstr” asks the manager to delete the original entity structure from, and add the renamed entity structure to, the entity list.

VI. ENTITY STRUCTURE BASE/MODEL

BASE MANAGEMENT

System entity structures represent structural knowledge about systems. The system entity structures are saved in an entity structure base (ENBASE) for later use. To do so, we store the entity structure created in the current Scheme environment in a directory called ENBASE on an external storage device such as a disk for later use.

A. Entity Structure Base

We have implemented ESP-Scheme so that it can save entity structures in, and retrieve them from, the ENBASE directory by using two facilities, “save-entstr” and “load-entstr.” The facility “save-entstr” saves an entity structure or a pruned entity structure into the ENBASE directory by storing a pair consisting of a list of items and a list of branches for the entity structure in the form of a disk file. A file name in the ENBASE directory, corresponding to the entity structure, is the same as its root entity name except for the extension of the file name, which can be either “.e” for the entity structure or “.p” for the pruned entity structure. The facility “load-entstr” searches for a
file corresponding to an entity structure in the ENBASE directory, retrieves the items list and branches list for the corresponding entity structure, and constructs the entity structure.

B. Entity Structure Manager

Entity structure manager (ESM) module manages all of the system entity structures in the current environment and/or in the ENBASE directory. The ESM module has three local variables. The first is a list of entity structures in the ENBASE or in the current environment. The second is a list of pruned entity structures in the ENBASE directory or in the current environment. The third is a list of both entity structures and pruned entity structures in the current environment. The operations in ESM include showing entity structures, adding entity structures, and deleting entity structures.

The operation "show-all" shows all entity structures in the current environment and/or in the ENBASE directory. The operation shows the entity structures and pruned entity structures in the three separate lists described in the preceding. The operation "add-entstr" adds an entity structure to the ESM list whenever an entity structure is created or renamed by the facilities described in Section V. The "delete-entstr" operation deletes an entity structure from the list in the ESM.

The initialization routine of ESP-Scheme initializes the ESM when ESP-Scheme is loaded. The initialization includes searching the ENBASE directory and building entity structures lists (described previously) so that the user may use the entity structures present in the current environment or retrieve some from the ENBASE directory as required. The current ENBASE directory can be moved from one place to the other as requested by the user by using "change-dir." Any change in the ENBASE directory results in the reinitialization of the ESM.

C. Transforming into DEVS Models

A pruned entity structure is synthesized into a simulation model by the transform operation. As transform visits each entity in the pruned entity structure, it calls upon a retrieval process that searches the model base (MBASE) directory for a model corresponding to the current entity. If one is found, it is used, and transformation of the entity subtree is aborted. The retrieval process proceeds by evaluating retrieval rules. The retrieval rules consist of condition/action productions and conflict resolution rules.

Prior to searching for a model, the name of the current entity is examined. If the current entity name is segmentable into a base name and a nontrivial extension (the extension must start with numbers or "&"), the base name is used as the entity name for the retrieval process.

One rule for searching for a model that corresponds to the current entity is to first look in the working memory, then in the MBASE directory, and finally, if the current entity is a leaf, in the ENBASE directory. If more than one rule condition is satisfied when evaluated, conflict resolution is used to fire only one rule. The conflict resolution strategy is context specificity—the rule with the most specific condition(s) is fired. Details of retrieval rules and conflict resolution rules are available in [13].

If a pruned entity structure is found in the ENBASE directory during the searching process, a transform is invoked and executed in a separate Scheme environment so as not to interfere with the current environment. Each recursive invocation can occur in a leaf entity only.

The integration of the ESP-Scheme model base management system into the DEVS-Scheme simulation environment is demonstrated by the following simulation study.

VII. SIMULATION OF THE REGISTRATION SYSTEM VIA PHONE (RSVP) SYSTEM AT THE UNIVERSITY OF ARIZONA USING DEVS-SCHEME

A registration system via phone (RSVP) has been recently installed at the University of Arizona. There have been many complaints and concerns as to whether the RSVP system in its current configuration is adequate to handle the needs of the University of Arizona. Additionally, there has been an expressed desire to incorporate some form of advertising such as student yearbooks, student health insurance, and others into the system [4].

The following provides an overview of the simulation study of the RSVP system using the discrete event system specification (DEVS)-Scheme object-oriented modeling and simulation environment described earlier.

A. Objective of the Simulation Study

The objective of the project was to assist the University Center for Computing and Information Technology by providing a realistic simulation of the existing RSVP system and then investigate the impact of adding advertising to the system.

B. Specific Issues Addressed by the Study

1) Number of telephone lines required to support the system. Currently there are 32 lines by which students can access the system. The University is considering a request to increase the number of lines to improve the system's student registration throughput.

2) Effects of the length of time to perform a transaction. This issue will address the effect of adding advertising onto the system after a student's transaction is completed.

C. Measures of Performance

The primary performance measures are the grade of service provided, utilization rate of the lines, and the student registration throughput. The grade of service is a ratio of call completions to call attempts. The utilization rate is a ratio of the time the lines are being used to the
total time available for use. The throughput is measured by the number of students who can be registered per unit of time.

VIII. RSVP System Description

The RSVP system consists of a Perceptron VOCOM-1 Package, an IBM 3725 communications processor which serves as a terminal controller, and an IBM 3090 mainframe computer. The VOCOM-1 package consists of two major components: a BT-II unit and a VOCOM unit. The BT-II unit, based upon a PDP-11 minicomputer with a real-time operating system, answers the phones (up to 32 at a time), provides the voice feedback to the student, and provides a translation from the touchtone phone inputs into an ASCII format. The VOCOM unit is primarily a VT-100 terminal with a tape drive which displays system operating status and statistics and provides a translation of the ASCII to EBCDIC. The IBM 3725 communications processor acts as a terminal controller and treats the VOCOM-1 package as an IBM 3271 terminal. The RSVP system software runs on the IBM 3090 mainframe computer. A block diagram of the system is shown in Fig. 9.

IX. RSVP Simulation Study Procedures

The experimental procedures followed during the course of this study were as follows: data collection, data analysis, development of assumptions, model construction, model validation, simulation experiments, and simulation output analysis. These are the same procedures that should be used in any simulation project. This simulation example will focus on the assumptions, model construction, model validation, simulation experimental procedures, and the simulation output analysis.

A. Assumptions

The assumptions made during the simulation study are as follows.

1) The response time of the IBM terminal controller and mainframe are not a consideration, since the system has submillisecond response time. Additionally, the system is designed to support approximately 5000 terminals in the transaction mode, and only 300 terminals are connected at the present time.

2) System failure was not considered, since there is only one system. Thus, if the system fails, it is the same as not having the system.

3) Single-line failure was also not considered, since there was not any available data indicating the individual line failure rate.

4) All calls will be at least 30 s in length. The BT-II does not detect a disconnection unless the call is terminated by entering the transaction termination code or times out. If a transaction termination code is not received within 15 s of the last input, the system will generate a message and then wait another 15 s. The system will then disconnect the line after the second 15-s wait if no input is received.

5) Each registration line simulated requires a separate random number stream. This provides a better representation of the randomness in the call length generation.

6) The data for calls being answered will be collected upon completion of the call rather than at the time the call was answered. This assumption will allow the reduction of messages passed between the models.

7) The call lengths will be determined by each line rather than by the call generator. This saves simulation run time, since a call length will not need to be determined for the call attempts that go unanswered.

8) Advertising will increase each call length by 15 s. The length of the advertising is based upon the experience of a broadcast engineer.

B. Model Construction

Model construction in the DEVS-Scheme/ESP-Scheme environment consists of three subactivities: specification of the model structure, specification of the model behavior, and synthesis of a simulation model [13]. The structure of the system is represented by an SES. The behavior of the system is represented by the models in the model base. The simulation model synthesis is accomplished via the transform operation described earlier.

1) Model Structure: To generate inputs to the RSVP system and to measure its performance, an experimental frame was used. An experimental frame is coupled model composed of atomic-models that are used to generate inputs, observe outputs, and provide control in accordance with the desired experimental conditions. The SES coupling the RSVP and experimental frame is shown in Fig. 10. There were two specializations for the controller implementation: broadcast and multicast. The broadcast specialization allows all registration lines to receive the input message. The multicast specialization specifically designates which registration line is to receive the input. The coupling schemes designated by (CS1), (CS2), and (CS3) in Fig. 10 show the external input, external output, and internal port connections of the models.

Our study focused on the performance of the BT-II registration system, with and without advertising and also with a varied number of registration lines. To obtain the particular model structure for simulation required the use
was constructed by selecting WITHOUT-TIME-LIMITS specialization from GEN-spec, the MULTICAST specialization from TRANS-TYPE-spec, and WITHOUT-TIME-LIMITS specialization from ADVERTISING specialization from RL-spec during the pruning operation. Our initial simulation model required the implementation of four atomic models. Two of the models were required for the experimental frame (GEN and TRANS), and two models were required for the RSVP system (RL and MULTI-CAST-CONTROLLER).

GEN and TRANS were required to generate the call attempts and to collect output data, respectively. The atomic model RL represents the behavior of a single telephone registration line and the atomic model MULTI-CAST-CONTROLLER represents the behavior of the BT-II when selecting an available telephone registration line. The DEVS-Scheme implementation of the atomic model RL without advertising is shown in Fig. 12. More detail on model construction and simulation can be found in [34].

The simulation of the registration procedure is initiated by the generator outputting a call attempt to the BT-II-RSVP system and TRANS. The BT-II-RSVP system outputs the call attempt to the MULTI-CAST-CONTROLLER, which in turn passes the call attempt to a registration line if available, otherwise the call attempt is ignored. When a call is completed, the registration line sends a call completion notice to the MULTI-CAST-CONTROLLER. The MULTI-CAST-CONTROLLER then passes the completion notice to the BT-II-RSVP system, which in turn sends the completion notice to EXP. EXP completes the simulation of one registration by sending the completion notice to the TRANS.

Each atomic model constructed was tested independently using the facilities provided by DEVS-Scheme prior of the pruning operation. Our initial simulation model was constructed by selecting WITHOUT-TIME-LIMITS specialization from GEN-spec, the MULTICAST specialization from TRANS-TYPE-spec, and WITHOUT-ADVERTISING specialization from the RL-spec during the pruning operation. The number of registration lines for a particular simulation model was specified during the pruning process. Fig. 11 illustrates the pruned SES.

2) Model Behavior: To model the behavior of the RSVP system required the implementation of four atomic models. Two of the models were required for the experimental frame (GEN and TRANS), and two models were required for the RSVP system (RL and MULTI-CAST-CONTROLLER).

GEN and TRANS were required to generate the call attempts and to collect output data, respectively. The atomic model RL represents the behavior of a single telephone registration line and the atomic model MULTI-CAST-CONTROLLER represents the behavior of the BT-II when selecting an available telephone registration line. The DEVS-Scheme implementation of the atomic model RL without advertising is shown in Fig. 12. More detail on model construction and simulation can be found in [34].

The simulation of the registration procedure is initiated by the generator outputting a call attempt to the BT-II-RSVP system and TRANS. The BT-II-RSVP system outputs the call attempt to the MULTI-CAST-CONTROLLER, which in turn passes the call attempt to a registration line if available, otherwise the call attempt is ignored. When a call is completed, the registration line sends a call completion notice to the MULTI-CAST-CONTROLLER. The MULTI-CAST-CONTROLLER then passes the completion notice to the BT-II-RSVP system, which in turn sends the completion notice to EXP. EXP completes the simulation of one registration by sending the completion notice to the TRANS.

Each atomic model constructed was tested independently using the facilities provided by DEVS-Scheme prior
to storing the atomic model in the model base. This facilitates the debugging of the atomic models and any coupled model of which they are a component.

3) *Simulation Model Synthesis:* The simulation model of the RSVP system was synthesized using the transform operation described in Section VI-C of this paper. The transformed RSVP system model is as shown in Fig. 13.

C. Model Validation

The models were validated by comparing the output generated using 23 lines, with a stationary Poisson arrival rate of 396 attempts per hour, the call lengths normally distributed \( m = 3.0667 \text{ min}, s = 0.55375 \text{ min} \) to the predicted data developed by U.S. West Communications for CCIT. The attempts per hour and mean call length used for the validation run were the same attempts per hour and mean call length used by U.S. West in their calculations. The standard deviation was obtained from analyzing the data from August 12-27, 1988. This data included the data used by U.S. West to calculate the predicted performance of the RSVP system. The predicted grade of service was 90%; our simulation runs had an average grade of service of 87.35%.

The simulation runs were also replicated with 32 lines, using the same arrival rate and call lengths and compared to actual system data. The actual system performance indicated that the grade of service was 100%. Our simulation runs had an average grade of service of 97.95%. Simulation output for two of the validation runs is shown in Fig. 14.

If the initial 30 minutes are discarded, the grade of service for 23 lines and 32 lines is 89.04% and 100%. The low completion rates for the first 30 minutes is a result of collecting call-answered data upon completion of the call rather than at the time the call was answered.

In either case, the difference between the simulation output, the predicted performance, and actual system performance was less than 3%. Thus our model is considered to be a valid model of the University of Arizona RSVP system.

D. Simulation Experimental Procedure

The call attempt rate was modeled using a nonstationary Poisson process to reflect the change in the call attempts per hour. The simulation experimental procedure consisted of replicating each simulation run a minimum of three times from 7:00 AM to 10:00 AM. This allowed the system to reach the maximum call attempt rate, thus testing the system under maximum load. The call lengths were modeled using a normal distribution with \( m = 3.56945 \text{ and } s = 1.23502 \text{ minutes} \).

Four different system configurations were simulated: the current 32-line system, a 32-line system with advertising, a 64-line system without advertising, and a 64-line system with advertising.

Each replication of a particular configuration used a different set of random number streams. However, each configuration used the same set of random number streams for performance comparison purposes.

E. Simulation Output Data

A representative sample of the simulation data obtained from the RSVP model is as shown in Fig. 15. Data
was collected at 30-min intervals for a simulation period of 180 minutes for each replication.

F. Output Data Analysis

The current 32-line configuration of the RSVP system exhibited an acceptable grade of service between 60 and 90 min of simulation time and then deteriorated rapidly as the call attempt rate began to increase. When a 15-s advertisement was added to the system, the grade of service deteriorated from a high of 98% at 60 min of simulation to a low point of 77% at 180 minutes.

G. RSVP Simulation Conclusions and Recommendations

The current system configuration is not adequate to meet the current registration demand as it did not sustain a grade of service above 90%. If advertising is to be added to the current 32-line configuration, the grade of service will deteriorate even more. The 64-line configuration had a grade of service above 90%, and if advertising were to be added, there would be no appreciable change in the grade of service.

X. Conclusion

We have described an implementation of system entity structure in Scheme, ESP-Scheme, which serves as a model base management system for DEVS models. Also, we illustrated the use of these concepts by modeling and simulating the University of Arizona registration system via phone (RSVP).

The utility of ESP-Scheme also has been demonstrated in the construction of several complex hierarchical models for computer networks [24], advanced computer architectures [13], and multirobot systems [30], [34].

The system entity structure construct has thus been shown to provide a workable foundation for model base management in simulation environments and workbenches. This is the first demonstration, to the authors' knowledge, of a knowledge-based system that addresses the needs of a model repository as expressed in [9], i.e., to provide a sharable repository of models and a means of assisting users to synthesize models to satisfy the objectives of the current analytic task. ESP-Scheme provides a system-theory-based structuring of the model base and a goal-based means of constructing models from reusable components. The hierarchical, modular simulation modeling capability of the underlying DEVS-Scheme environment is critical to the workability of this approach.

The aforementioned DEVS-Scheme/ESP-Scheme knowledge-based framework serves as our vehicle for research in knowledge-based system design using variant families of design models [19]–[22].

Research is continuing to further explore, and respond to, the needs of model base management. Currently, our efforts are concentrating on maintaining model base coherence in the face of the multiplicity of related models expressed in various formalisms at various levels of abstraction.

REFERENCES

[22] ___, "A conceptual basis for model-based system design," Ph.D.
dissertation, Dept. of Computer Science, Wayne State University, Detroit, MI, 1985.


Tag Gon Kim (S’84–M’88) received the B.S.E.E. and M.S.E.E. degrees from Pusan National University, Korea, and Kyungpook National University, Korea, in 1975 and 1980, respectively. He received the Ph.D. degree in electrical engineering from the University of Arizona, Tucson, AZ, in 1988. From 1987 to 1989, he worked as a Research Staff Engineer in the Environmental Research Lab of the University of Arizona. Since August 1989, he has been an Assistant Professor in the Telecommunications and Information Sciences Lab, Department of Electrical and Computer Engineering, The University of Kansas, Lawrence, KS. He also held a faculty position in the Department of Electronics, National Fisheries University of Pusan, Pusan, Korea. His research interests include artificial intelligence for advanced simulation methodology, computer systems modeling, and object-oriented software environment.

Dr. Kim is a member of the ACM, the AAAI, the SCS, and Eta Kappa Nu.

Chilgee Lee received the M.S.E.E. degree from Arizona State University and the B.S.E.E. degree from Sungkyunkwan University, Seoul, Korea. He is a Ph.D. candidate in the Artificial Intelligence and Simulation Group, Department of Electrical and Computer Engineering, at the University of Arizona.

His research interests are in the areas of artificial intelligence, modeling and simulation, computer architectures, and parallel processing.

Eric R. Christensen (M’83) received the M.S.E.E. degree from the Air Force Institute of Technology, and the B.S. degree from the United States Military Academy.

He is a Major in the U.S. Army Signal Corps and a Ph.D. candidate in the AI and Simulation Group, Department of Electrical and Computer Engineering, University of Arizona. His research interests are in the areas of artificial intelligence, modeling and simulation, command, control, and communications systems design.

Mr. Christensen is a member of the AFCEA.

Bernard P. Zeigler received the Ph.D. degree in computer/communication science from the University of Michigan, Ann Arbor, in 1969, and the preceding degrees from McGill University, Montreal, PQ, Canada, and from MIT, Cambridge.

He is a Professor in the Department of Electrical and Computer Engineering at the University of Arizona. He is the author of Multi-facetted Modelling and Discrete Event Simulation (Academic Press, 1984) and Theory of Modelling and Simulation (Wiley, 1976). His research interests include artificial intelligence, distributed simulation, and expert systems for simulation methodology.