ESP-scheme: A realization of system entity structure in a LISP environment

Tag Gon Kim, ERLab and B.P. Zeigler, ECE Dept.

University of Arizona
Tucson, AZ 85721

ABSTRACT

System entity structure (SES) developed by Zeigler is a structural knowledge representation scheme that contains knowledge of decomposition, taxonomy, and coupling of a system. Formally, SES is a labeled tree with attached variable type that satisfies certain axioms. This paper describes a realization of SES in Scheme (a LISP dialect) called ESP-Scheme. The paper first presents representation of SES and main operations on SES, and then describes facilities in ESP-Scheme. ESP-Scheme acts as a model base management system in DEVS-Scheme, a knowledge-based simulation environment developed by the authors. It supports specification of the structure of a family of models, pruning the structure to a reduced model, and transforming it to a simulation model by synthesizing components models in the model base.

1. Introduction

DEVS-Scheme realizes Zeigler's *DEVS* (Discrete Event System Specification) formalism in Scheme (a LISP dialect) (Kim and Zeigler, 1987; Kim, 1988; Zeigler, 1987). The environment supports building models in a hierarchical, modular manner, a systems oriented approach not possible in conventional languages (Concepcion and Zeigler, 1988). To organize such 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, 1984) is one such tool for the model base management. ESP-Scheme is a realization of the system entity structuring formalism in the Scheme environment. The ESP-Scheme supports hierarchical specification of the structure of a family of models, pruning the structure to a reduced model, and transforming the structure to a simulation model by synthesizing components models in the model base developed using DEVS-Scheme. This paper first reviews the system entity structuring formalism and then describes 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 modelling and simulation based on the entity structure and model base.

2. System Entity Structuring Formalism: A Review

A *system entity structure* (*SES*) is a knowledge representation scheme which contains the decomposition, coupling, and taxonomy information necessary to direct model synthesis (Zeigler 1984; Rozenblit, 1985). Formally, the *SES* is a labeled tree with attached variable types that satisfies five axioms—alternating mode, uniformity, strict hierarchy, valid brothers, and attached variables. Detail description of the axioms is available in (Zeigler, 1984).

2.1 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. 1) 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 *specialization* node (a double vertical arrows in the labeled tree of Fig. 1) represents a way in which a *general* entity can be categorized into *special* entities. As shown in Fig. 1, 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 homogeneous 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. 1). The representation of such a multiple entity is as follows. A multiple entity A and its components Bs are represented by A, three vertical lines, and B from the top down. Note that instead of presenting all Bs for A's components, only one B is placed in the labeled tree. The number of Bs is specified by a variable, which is attached to the multiple aspect node.

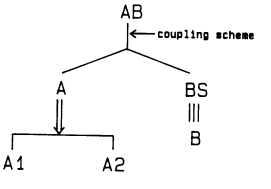


Fig. 1. A System Entity Structure.

2.2 Operations on SES

A *SES* represented by a labeled tree consists of branches and nodes (nodes are also called *items*). An *item* in the *SES* is in one of three types—entities, aspects, or specializations—an ordered pair of which is represented by a branch of the *SES*.

Some operations 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*, transforming the pruned *SES* into a model.

As the construction of a *SES* is a sequence of adding new items – entities, aspects or specializations – to the *SES*, the most common operation is adding an item. The deletion operation, which deletes entities and associated branches from the *SES*, can be applied only to those entities with no aspects. Variables can be attached to and removed from the items in the *SES*.

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 a model. The transform operation synthesizes a model in a hierarchical fashion from components in the model base developed by using the DEVS-Scheme.

3. Implementation of ESP-Scheme

3.1 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. 2. 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, mult-coup-type*, and *attributes-lst*. Each branch in the *branches-list* is represented by another structure type called *branch* that maintains an *ordered pair* of two items, left- and right-items, in the SES.

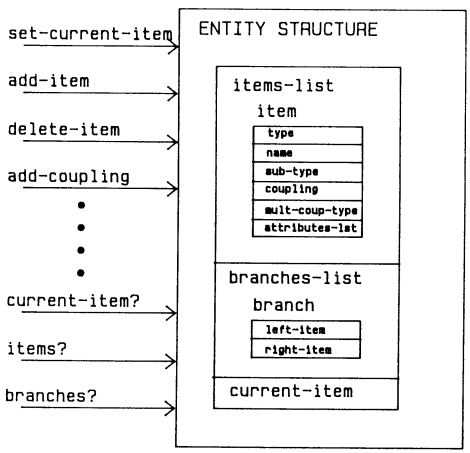


Fig. 2. Representation of System Entity Structure Module.

The field *type* in item structure represents the type of an entity in an entity structure whose range is in {entity, aspect, specialization}. The field *name* is used to identify an entity by its name. The field *coupling* is used to specify 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 coupling scheme is compatible with that of DEVS-Scheme. The field *sub-type* with range {multiple-entity, multiple-aspect, multiple-children} represents information on multiple entities and multiple decomposition. The field *mult-coup-type* with range {broadcast, hypercube, cellular} is used to specify the coupling scheme for a kernel model in DEVS-Scheme. Items in *SES* may have attributes that characterize their features. The field *attributes-lst* maintains a list of such attributes, each of which is a pair of variable and its value.

The main operations on the *SES* are *set-current-item, add-item, add-mult, delete-item, add-coupling, prune*, and *transform*. To explain such operations, let us build the *SES* of a system BUF-PROC, a processing element containing a buffer cascaded with a processor. The type of buffer, assumed to be either FIFO (First In First Out) or LIFO (Last In First Out), will be selected by the user in the pruning process. Once the *SES* of the BUF-PROC with an aspect and a specialization is built, we prune the BUF-PROC entity structure and transform the pruned BUF-PROC in a model.

Building the *SES* for the BUF-PROC starts with creating an entity structure, the root of which is an entity BUF-PROC (line (1) of Fig. 3 (a)). 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 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. 3 (a)). To add other items under the aspect *comp-dec* requires setting the current item to the aspect *comp-dec* (line (3) of Fig. 3 (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. 3 (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. 3 (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*.

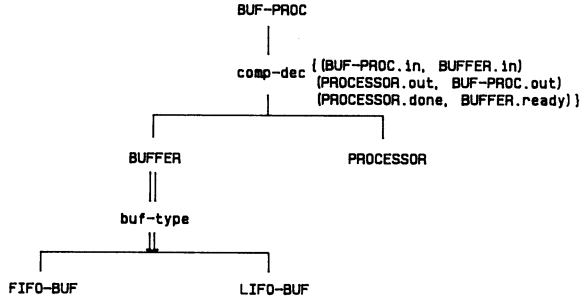
The coupling scheme of the BUF-PROC system, which is attached to the aspect *comp-dec*, can be specified by the operation *add-coupling*. This operation needs to specify the names of two entities and the names of ports in the two entities. The operation *add-coupling* specifies both internal and external 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. 3 (a) specify the internal coupling scheme of the BUF-PROC.

Fig. 3 (b) shows the resulting *SES* for the BUF-PROC system, which has one aspect under the BUF-PROC entity and one specialization under the BUFFER entity.

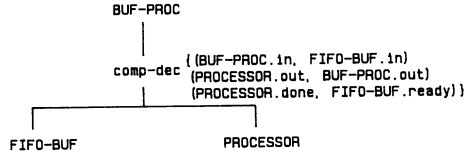
Having specified the *SES*, we are ready to prune it to select a particular *pruned entity structure*. The pruning operation queries the user to select one entity under a specialization, if there is one, while traversing all items in the *SES*. One such pruned entity structure is shown in Fig. 3 (c), where the FIFO-BUF has been selected as a specialized buffer.

```
(1)  (make-entstr 'BUF-PROC)
(2)  (add-item e:buf-proc 'asp 'comp-dec)
(3)  (set-current-item e:buf-proc 'comp-dec)
(4)       (add-item e:buf-proc 'ent 'BUFFER)
(5)       (add-item e:buf-proc 'ent 'PROCESSOR)
(6)       (set-current-item e:buf-proc 'BUFFER)
(7)            (add-item e:buf-proc 'spec 'buf-type)
(8)            (set-current-item e:buf-proc 'buf-type)
(9)                 (add-item e:buf-proc 'ent 'FIFO-BUF)
(10)                (add-item e:buf-proc 'ent 'LIFO-BUF)
(11) (set-current-item e:buf-proc 'comp-dec)
(12) (add-couple e:buf-proc 'BUF-PROC 'BUFFER 'in 'in)
(13) (add-couple e:buf-proc 'PROCESSOR 'BUF-PROC 'out 'out)
(14) (add-couple e:buf-proc 'BUFFER 'PROCESSOR 'out 'in)
(15) (add-couple e:buf-proc 'PROCESSOR 'BUFFER 'done 'ready)
```

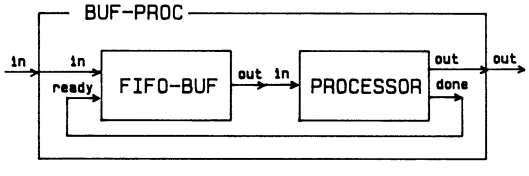
(a)



(b)



(c)



(d)

Fig. 3. (a) ESP-Scheme Code (b) System Entity Structure
(c) Pruned Entity Structure (d) Transformed Model

To automatically construct a simulation model in DEVS-Scheme we apply the *transform* operation to a pruned entity structdure. The operation *transform* retrieves the models from the model base, which correspond to the entities in the pruned entity structure, and then synthesizes them into a simulation model for the BUF-PROC (Fig. 3 (d)). For such models to be available in the model base, we need to specify *atomic DEVS* models for the FIFO-BUF, LIFO-BUF, and PROCESSOR and save them in the model base before applying the transform. Details of the transform operation will be described in section 5.3

3.2 Multiple Entity

The specification of multiple entity and multiple decomposition is powerful for specifying a massively parallel computer architecture with appropriate connection topologies—such as broadcast, hypercube, or cellular—which has been recently introduced as a special-purpose architecture. To demonstrate the power of such specification of the parallel architecture, let us assume that the parallel processor has a collection of processing elements with one of three connections above, each element of which is the BUF-PROC that has already been specified. Let us call the parallel processor BUF-PROCS, meaning that BUF-PROCS consists of a collection of BUF-PROCs.

We can build the *SES* for the BUF-PROCS by adding extra specifications to the one shown in Fig. 3 (a) to create a new root entity BUF-PROCS of type multiple entity. 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 (2) through (15) of Fig. 3 (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 constrast to *add-coupling* for digraph models. It sets the slot *mult-coup-type* of the item of type *multiple aspect* to one of subclasses of kernel models (Kim, 1988) in DEVS-Scheme. The resulting *SES*, which has a multiple entity, is shown in Fig. 4. If the broadcast coupling is specified for the multiple decomposition in pruning process, the operation *transform* will create a broadcast model for the BUF-PROCS consisting of one atomic BUF-PROC model. The number of components, BUF-PROCs, can be specified in the initialization process before simulation is begun.

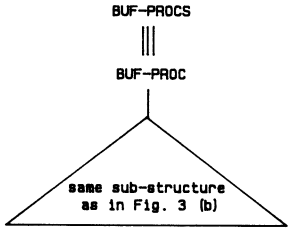


Fig. 4. System Entity Structure with Multiple Entity.

3.3 Hierarchical Model Structuring Operations

To show the power of *SES* hierarchical models structuring formalism we will discuss some advanced operations. Others can be found in (Kim, 1988).

The operation *add-item* is extended to the operation *add-sub-entstr*. This operation adds one 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* needs to specify an item so that a subentity structure consisting of all items under the specified item will be deleted.

The operation *add-mult* is extended such that the level of hierarchy for the multiple entities to be added is arbitrary. The operation *add-mult-mult* allows us to specify a hierarchical construction of different kernel models. Fig. 5 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 modelling a multi-level hypercube architecture can be found in (Kim and Zeigler, 1988). The operation *attach-num-mult-children* attaches the number of components under a multiple entity to the multiple entity.

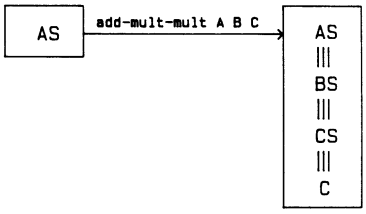


Fig. 5. Construction of Multi-level Multiple Entities.

3.4 Reuse of Pruned Entity Structures

The *Entity Structure Module* provides several operations for reuse of 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. 6). 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 non-leaf entity into a leaf entity by cutting all entities under the non-leaf entity, constructing a new entity structure with the entities cut, and saving it in the entity structure base (we shall describe entity strcuture base in section 5.1). The root of the new entity structure is the leaf entity under which all cut entities are connected with the same structure that they used to be. After creating the new entity structure, the operation asks the user to prune the created entity structure as many times as desired. The pruned entity structures are saved in the entity structure base for later use by the operation *add-spec&ents-at-leaf*. Fig. 7 shows the operation *cut-entstr*.

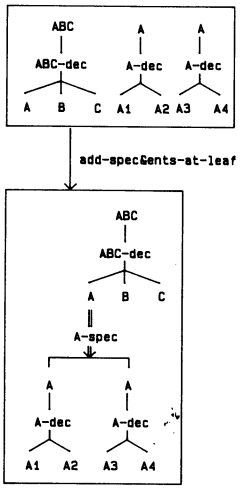


Fig. 6. Addition of Pruned Entity Structure.

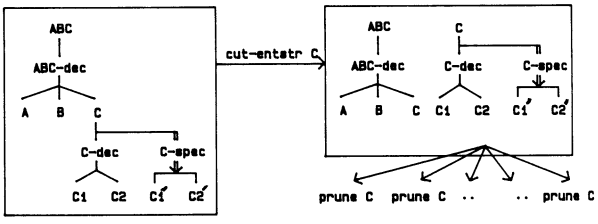


Fig. 7. Cutting Sub Entity Structure.

An entity structure can be constructed in a hierarchically distributed manner so that its leaf entities have their respective entity structures in the entity structure base. The entity structure so built may be merged into one entity structure. The operation *merge-entstr* searches entity structures in the entity structure base with the same root names as leaf entities of an entity structure. If such entity structures are present in the entity structure base, they replace the respective leaf entities.

4. Facilities in ESP-Scheme

SES construction, copying, and other facilities are provided by ESP-Scheme. 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 so created has only the root entity system, items should 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, the facility create an entity structure that has the same structure as the original one. The facility copies a list of items and a list of branches from the original entity structure and constructs a new entity structure.

Since a list of all entity structures is maintained by the entity structure manager (described in section 5.2), any facility that creates or deletes entity structures should report to the manager about the creation and/or deletion of entity structures so that the manager can update the list of entity structures within it. For example, the facility *rename-entstr* asks the manager to delete the original entity structure from, and add the renamed entity structure to, the list within the manager.

5. Entity Structure Base/Model Base Management

The system entity structures represent the structure knowledge about systems. Such system entity structures are to be saved in the *entity structure base* (*ENBASE*) for later use. To do so, we save an entity structure created in the current Scheme environment into an external storage such as a disk for later use.

5.1 Entity Structure Base

We have implemented ESP-Scheme such that it can save entity structures in, and retrieve them from, the *ENBASE* by using two new facilities, *save-entstr* and *load-entstr*. The facility *save-entstr* saves an entity structure or a pruned entity structure into the *ENBASE* 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*, corresponding to an 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*, retrieves the items list and branches list for the corresponding entity structure, and constructs the entity structure.

5.2 Entity Structure Manager

A module *entity structure manager* (*ESM*) is designed to manage all system entity structures in the current environment and/or in the *ENBASE*. The module *ESM* has three local variables. The first variable is a list of entity structures either in the *ENBASE* or in the current environment. The second variable is a list of pruned entity structures either in the *ENBASE* or in the current environment. The third variable is a list of both entity structures and pruned entity structures in the current environment. The operations on the *ESM* include: show entity structures, add entity structure, and delete entity structure.

The operation *show-all:ens* shows all entity structures in the current environment and/or in the *ENBASE*. The operation shows the entity structures and pruned entity structures in the three sperate lists, as described above. The operation add entity structure adds an entity structure to the list in the *ESM* whenever the *ESM* receives a report on the creation of the entity structure from the facilities as described in section 4. The delete entity structure operation deletes an entity structure from the list in the *ESM* as required by the facilities *delete-entstr* or *rename-entstr*.

The initialization routine of the ESP-Scheme initializes the *ESM* when the ESP-Scheme is loaded. The initialization includes searching the *ENBASE* and setting up all lists of entity structures in the *ESM* so that the user can get available entity structures and retrieve some as required. The current *ENBASE* can be moved from one place to the other as requested by the user using *change-dir*. Any change in the *ENBASE* results in the reinitialization of the *ESM*.

5.3 Transform into DEVS Models

A *pruned entity structure* can be synthesized into a simulation model by the operation *transform*. As it visits each entity in the pruned entity structure, *transform* calls upon a *retrieval process* that searches 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 the *retrieval rules*, which consist of retrieval rules—pairs of *condition* and retrieval *action*—and *conflict resolution rules* by which a rule is selected if there is more than one with conditions satisfied.

One rule for searching a model that corresponds to the current entity first looks for the model in the working memory, then in the MBASE, and finally, if the current entity is a leaf, in the *ENBASE*. Before searching the model, another rule checks the name of the current entity. If the current entity has a base name and a *non-trivial* extension (the extension starts with numbers or "&"), the base name is used as an entity name for the retrieval process. If more than one rule is satisfied when evaluated, a conflict resolution rule fires only one rule. We employ context specificity—the rule with a more specific condition than other rules is fired—to resolve such a conflict. Details of retrieval rules and conflict resolution rules are available in (Kim, 1988).

If a pruned entity structure is found in the *ENBASE* in 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.

6. Summary and Discussion

We have described an implementation of system entity structure in Scheme, ESP-Scheme, which serves as a model base management system for DEVS models. Our experience with the model management system has shown that it is easier to specify coupled models by defining the appropriate entity structure and applying transform for model synthesis. The transform requires existence of atomic models in the model base defined using DEVS-Scheme.

The utility of ESP-Schem has been demonstrated in construction of several complex hierarchical models for computer networks (Sevinc and Zeigler, 1988) and advanced computer architectures (Kim, 1988).

The above knowledge base framework, implemented in (Kim, 1988), intended to be generative in nature. It is a compact representation scheme which can be unfolded to generate the family of all possible models synthesizable from components in the model base. This knowledge base framework

serves as our vehicle for research in knowledge-based system design using variant families of design models (Rozenblit and Zeigler, 1985; Rozenblit, 1986; Sevinc and Zeigler, 1988).

REFERENCES

Concepcion, A.I. and B.P. Zeigler "DEVS Formalism: A Framework for Hierarchical Model Development," *IEEE Trans. Software Engr.*, vol. SE-14, no. 2, pp. 228-241, Feb. 1988.

Kim, Tag Gon "A Knowledge-Based Environment for Hierarchical Modelling and Simulation," Ph.D. Dissertation. Dept. of Electrical and Computer Engr., University of Arizona, Tucson, AZ, 1988.

Kim, Tag Gon and B.P. Zeigler "The DEVS Formalism: Hierarchical, Modular System Specification in an Object-Oriented Framework," In *Proc. of 1987 Winter Simulation Conf.*, Atlanta, GA, 1987, pp. 559-566.

Kim, Tag Gon and B.P. Zeigler "The Class Kernel-Models in DEVS-Scheme: A Hypercube Architecture Example," *ACM SIMULETTER*, vol. 19, no. 2, June, 1988.

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

Rozenblit, J.W. and Zeigler, B.P. "Concepts of Knowledge Based System Design Environments," In *Proc. 1985 Winter Simulation Conf.*, San Francisco, CA, 1985, pp. 223-231.

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

Sevinc S. and B.P. Zeigler "Entity Structure Based Design Methodology: A LAN Protocol Example," *IEEE Trans. Software Engr.*, vol. SE-14, no. 3, pp. 375-383, Mar. 1988.

Zeigler, B.P. *Multifacetted Modelling and Discrete Event Simulation.* London, UK and Orlando, FL: Academic Press, 1984.

Zeigler, B.P. "Hierarchical, Modular Discrete-Event Modelling in an Object-Oriented Environment," *Simulation*, vol, 50, no. 5, pp. 219-230, 1987.

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 Electrical and Computer Engineering Department 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