Entity structure management of continuous simulation models

Tag Gon Kim
Guoqing Zhang
Bernard P. Zeigler

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

ABSTRACT

The systems entity structure knowledge representation framework underlies DEVS-SCHEME, a general purpose environment for constructing hierarchical discrete event models in an object-oriented environment. In this paper, we show how DEVS-SCHEME also serves as an interface to existing simulation systems. To illustrate the entity structure concepts, we show how such an interface organizes the modules of a continuous simulation package called TRNSYS for which an extensive set of routines for greenhouse micro-climate modelling are available. Such routines are selected, modified if needed, and coupled together by the modeller to meet his current objectives. DEVS-SCHEME writes a file containing such specifications to drive the TRNSYS simulation executive. The modeller can thus work with the entity structure pruning interface to construct a model rather than writing directly an extensive set of coupling specifications in an obscure numerical coding.

1. Introduction

The system entity structure (SES) is an enriched knowledge representation framework capable of utilizing knowledge representation schemes deriving from both AI approaches and from dynamic systems (Zeigler 1986). DEVS formalism (Zeigler 1984) offers the formal means to describe hierarchical and modular discrete event models. The DEVS-SCHEME (Zeigler 1987, Kim and Zeigler 1987), which implements the SES and DEVS formalisms, is a general environment for constructing, organizing and simulating hierarchical, modular discrete event models. By taking advantage of the general nature of the DEVS-SCHEME, an interface was built between the DEVS-SCHEME and a continuous simulation system, TRNSYS (1983). The interface utilizes the power in the DEVS-SCHEME to organize TRNSYS models by employing system entity structure facilities. To obviate having to start from scratch each time a model is needed, models may be kept in an organized library called MODEL-BASE. The system entity structure organizes models in such a model base so that they can readily be retrieved and (re)used at the command of the designer/decision-maker or his computerized assistant.

This paper will present how continuous system models can be uniformly organized by employing system entity structure facilities which are embedded in the DEVS-SCHEME environment. To clarify this approach, system entity structure concepts will be briefly described, followed by the description of TRNSYS simulation system. A TRNSYS system example is presented to demonstrate the effectiveness of the approach. Activities performed by the interface between the DEVS-SCHEME and TRNSYS are then discussed. In conclusion, some prospective research topics are brought up concerning the extension of the DEVS-SCHEME.

2. SES Management of Models

The DEVS-SCHEME is a realization of DEVS (Discrete Event System Specification) formalism in an object-oriented programming environment, SCOOPS of PC-SCHEME. The DEVS-SCHEME supports hierarchical, modular specification of discrete event models with simulation modelling and AI techniques combined. The system entity structuring formalism (Zeigler 1984) is a representation scheme of the structural knowledge of systems, which underlies the DEVS-SCHEME. ESP-SCHEME is a realization of the system entity structuring formalism in a LISP-Based, object-oriented programming environment. The ESP-SCHEME supports specification of the structure of a model, pruning the structure to the reduced one, and transformation the structure to a simulation model by synthesizing components models in the model base developed by using the DEVS-SCHEME.

2.1 System Entity Structure Concepts

A system entity structure (SES), a representation scheme of the structural knowledge of systems, is a labeled tree with attached variable types (Zeigler 1976, 1984). Zeigler and Zhang (1988) formalized SES using set theoretic mechanism. The formalization of SES makes it possible to prove the completeness, correctness and irredundancy of two kinds of transformations which support the use of the SES for automated construction of models from a model base. The two transformations will be described in later sections.

For more detailed discussion on the concepts of System Entity Structure the reader is referred to (Zeigler 1976, 1984)

2.2 Three Relationships in the 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 (eg. MODULE in Figure 1), having several aspects and/or specializations, corresponds to a model component that represents a real world object. The aspect node (eg. MODULE_DEC in Figure 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 (eg. SYSTEM_TYPE in Figure 1) represents a way in which a general entity can be categorized into special entities.

A multiple entity (eg. MODULEs in Figure 1) is a special entity that consists of a collection of homogeneous components (eg. MODULE in Figure 1). 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). Note that instead of presenting all components for the MODULEs, only one MODULE is placed in the labeled tree. The number of components is specified by a variable, which is attached to the multiple aspect node.

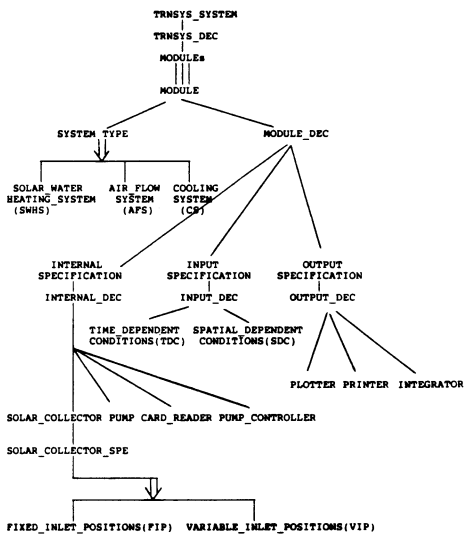

Figure 1. Simplified Entity Structure for TRNSYS_SYSTEM

2.3 Operation on System Entity Structure

The knowledge representation framework, SES, just introduced is intended to be generative in nature, i.e., it should be a compact representation scheme which can be unfolded to generate the family of all possible models synthesizable from components in the model base. Each final system structure generated above is represented by a pure entity structure, which has no specializations.

Pruning operations are performed in order to obtain a pure entity structure. These operations can then ultimately reduce the structure to a composition tree (Zeigler 1984) which contains all the information needed to synthesize a model in hierarchical fashion from components in the model base. Rozenblit (1985) proposed some algorithms for this purpose. Further elaboration on this approach can also be found in (Zeigler and Rozenblit 1985)

Two exclusive and complete forms of transformations, part of pruning operations, are presented by Zeigler and Zhang (1988). Ways are also proposed to prove the correctness of these transformations by building a formal definition for SES based on set theory. Transformation-I applies whenever there are specializations directly under a multiple decomposition while Transformation-II applies whenever specializations are separated by at least one aspect from an enclosing multiple decomposition. In Figure 1, Transformation-I can be applied to SYSTEM_TYPE, while Transformation-II is applicable to SOLAR_COLLECTOR_SPE.

In addition to two forms of transformations, some other pruning operations are used to select an aspect for each non-atomic entity, to select an entity from a specialization and to concretize information associated with the aspects, entities and specializations. Here atomic entity is defined as having neither specialization nor aspect. As in Figure-1, PUMP is an atomic entity, but MODULE is not.

The operations on the SES are very useful, as we shall show, for users who can work with SES pruning interface to construct a continuous system model structure rather than writing an extensive set of coupling specifications for the system in an obscure numerical coding.

2.4 Model Management

The descriptions of the models in the model base must be in the proper modular form : explicitly expressed input and output ports through which the model must communicate with other models. Such modular components can be assembled in hierarchical fashion in a manner well captured by system entity structure specification. In the DEVS-SCHEME environment, models in the model base can be of either discrete or continuous types as long as they are recognizable by the interface to DEVS-SCHEME.

The management of continuous models, particularly the TRNSYS models, is the main topic of this paper. Figure-2 depicts the kinds of activities supported by the DEVS-SCHEME interface to TRNSYS. Such activities include (1) definition and modification of TRNSYS model types, (2) specification of coupling relations (system configurations), (3) automatic generation of system entity structure for a given TRNSYS system, (4) creation of

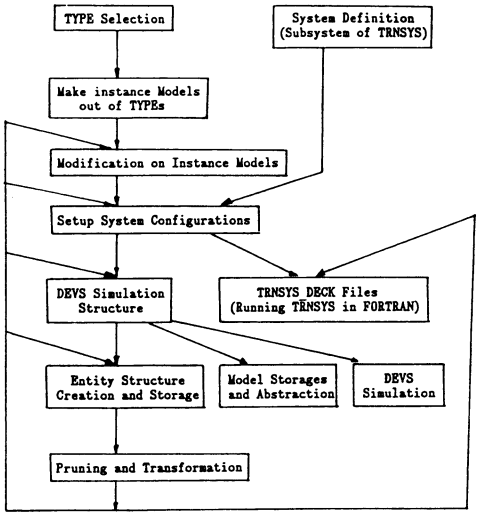

Figure 2. Activities for Model Management

TRNSYS executive files, and (5) for purpose of organizing continuous models, developing discrete event models corresponding to continuous models in TRNSYS model base.

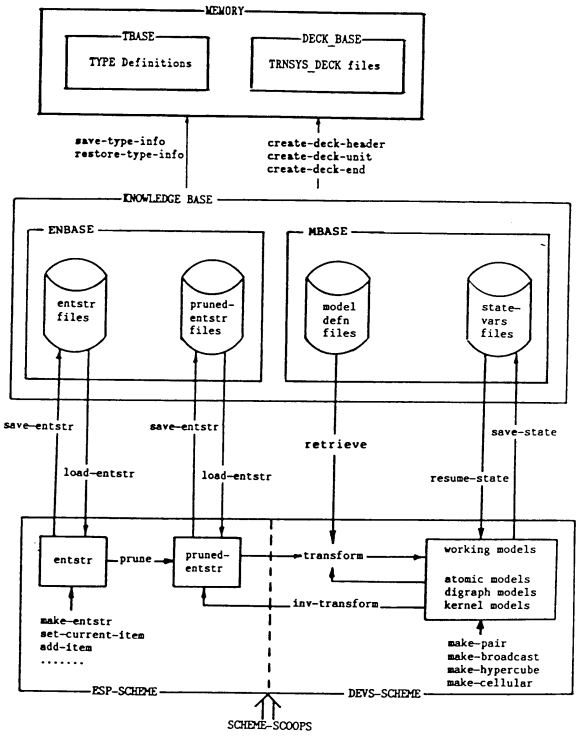

Figure 3. Environment of Model Management

Figure-3 shows the system composition for model management. In addition to the model base (MBASE) which stores the executable models in the DEVS-SCHEME environment, there are other three bases : (1) TYPE BASE (TBASE) which contains all information about TRNSYS model types, (2) System Entity Structure Base (ENBASE) which stores system entity structures facilitating organization of models, and (3) DECK BASE (DBASE) which has the information files necessary for TRNSYS to drive the simulations on the models.

The system entity structure in Figure-1 is stored in ENBASE. The atomic models (e.g. PUMP) are primitive ones for performing simulation tasks and are stored in MBASE. Some other coupled models, which are constructed hierarchically from other hierarchical models, also reside in MBASE. PUMP, in TRNSYS terms, is a model type whose information is kept in TBASE and shown in Figure 4. Figure 4 shows that any instance of type PUMP will have the type number 3, one parameter named MAXIMUM_FLOW_RATE with default value 200, three input and three output variables (ports), and some other control information. Knowledge about a TRNSYS system, Solar Water Heating System as in Figure 1, is stored in DBASE and is available to TRNSYS executives for simulation. Also included in the DBASE is the control information necessary for simulation runs, such as the simulation starting time.

```
TYPE NAME : PUMP
TYPE NUMBER : 3
NUMBER OF PARAMETERS 1
((MAXIMUM_FLOW_RATE 200))
NUMBER OF INPUTS 3
((INLET_FLUID_TEMPERATURE 20) (INLET_MASS_FLOW_RATE 200)
(CONTROL_FUNCTION ZERO))
NUMBER OF DERIVATIVES 0
()
NUMBER OF OUTPUTS 3
((OUTLET_TEMPERATURE ?) (OUTLET_MASS_FLOW_RATE ?)
(PUMP_POWER_CONSUMPTION ?))
DELETE-FLAGS
(() () () ())
ADDING-FLAGS
(() () () ())
```

Figure 4. Model Type : PUMP

Having addressed the fundamental concepts and structure of model management using the system entity structure framework, the TRNSYS system and its interface to DEVS-SCHEME will be given in the following sections.

3. Continuous Simulation System : TRNSYS

TRNSYS was developed at the University of Wisconsin at Madison (1983). It may simulate a variety of dynamic systems, and is especially well suited for the analysis of energy systems (e.g., heating, cooling and dehumidification systems for control of temperature and humidity). TRNSYS requires its users to describe the system to be simulated by means of an input data file coded in a special form. The file is referred to as TRNSYS_DECK. System components are defined in modular forms and the

interconnections among the components are set up by specifying the port-port relations. This results in the specification of a modular system model. Therefore, as a side product, one level of model abstraction/aggregation is made possible to some extent.

A TRNSYS system contains a set of interconnected components. For example, a typical and simplified solar water heating system (Figure 1) may consist of a solar collector, a pump, several temperature sensing controllers and some output equipment.

Each system component is an instance of certain so-called TRNSYS type identified with a Type Number. The system component itself has an unique unit number as the identifier.

Any TRNSYS system component may have parameters, input and output variables all of which are ordered sequentially with respect to their own categories (parameter, input or output). For example, the coupling relation (m,n) associated with the Ith input variable of the system component (with unit number j) tells that the component will receive information from another component (with unit number m) via its nth output port (variable).

A TRNSYS system can be constructed after the TRNSYS types are defined and stored in TBASE. Figure 5 shows the actual deck information associated with the system component PUMP. From Figure 5, it can be seen that PUMP belongs to type 3 and has 3 as its unit number. One parameter, three output and three input variables are defined for PUMP. These pieces of information are innerited from the type PUMP. The first two input variables (ports) receive no information from other system components during the simulation since (0,0)s are specified. The third input port, however, will be connected to the first output port of the system component with unit number 2. Three input variables have the initial values of 20, 200 and 0, respectively.

```
UNIT 3 TYPE 3 PUMP

     PARAMETERS 1
     200

     INPUTS 3
     0,0 0,0 2,1
     20  200 0

*** OUTPUTS 3
* OUTLET_TEMPERATURE OUTLET_MASS_FLOW_RATE
* PUMP_POWER_CONSUMPTION
```

Figure 5. DECK Data for PUMP

Comparison of TRNSYS system components with the entities in a system entity structure reveals many similarities. This in part motivates the setting up of the interface between TRNSYS and DEVS-SCHEME. As a consequence, the organization of continuous models is made possible via system entity structures facilities in the DEVS-SCHEME environment.

4. Model Management Environment and Activities

In the above sections, we have presented the TRNSYS type management by describing what is included in type definition (shown in Figure 4) and the TRNSYS_DECK format by presenting the deck information for PUMP, a system component of SWHS.

To illustrate how the models for solar water heating system (SWHS) are organized using the system entity structure framework, the following activities are conducted.

ACTION 1. Define/Modify TYPE information

This action defines all relevant type information in order to construct SWHS system components. By invoking (DEFINE-TYPE) in DEVS- SCHEME, all type information can be interactively keyed into TBASE. The necessary types for SWHS are : (CARD 1), (CONTROLLER 2), (PUMP 3), (COLLECTOR 1), (INTEGRATOR 24), (SOLAR 16), (PRINTER 25), AND (PLOTTER 26). The number inside each list is the type number for the type. For example, (PUMP 3) tells the type PUMP has the type number 3. In the above definitions, CARD means CARD READER, CONTROLLER means PUMP CONTROLLER, COLLECTOR means FLAT PLAT CONTROLLER, and SOLAR means SOLAR RADIATION PROCESSOR.

By entering (UPDATE-TYPE), users can modify the previous defined types interactively.

ACTION 2. Define TRNSYS System

Command (MK-SYSTEM SWHS) defines a TRNSYS system SWHS in DEVS- SCHEME environment. The following actions will set up the configuration of SWHS components.

ACTION 3. Choose all TYPEs Needed for SWHS

By typing (CHOOSE-TYPE PUMP PRINTER PLOTTER .....), the information about all types necessary to SWHS is extracted from TBASE to DEVS-SCHEME environment. As we noticed in Figure 4, some control information is also associated with the types. This allows the error checking during the construction of TRNSYS models.

ACTION 4. Define System Components

The command (DEFINE-INSTANCES PUMP 1) makes an instance of type PUMP as a system component of SWHS. Other system components can be similarly made for SWHS. Another example is (DEFINE-INSTANCES PRINTER 2) makes two printers for SWHS. The unit numbers are created internally by the interface in the order that the components are made.

ACTION 5. Define Coupling Specification

The coupling relations among the system components are defined interac-

tively using the command (CONSTRUCT-COUPLING SWHS). For each input port of each system component, the coupling relations with other components' output ports will be requested.

ACTION 6. Generate TRNSYS_DECK File

TRNSYS deck file for SWHS can now be generated with the command (CREATE-DECK SWHS). Users will be asked for some information in order to drive simulations in TRNSYS, e.g., start simulation time, etc.

ACTION 7. Create System Entity Structure

(CREATE-ENT) will create the system entity structure for SWHS and store it in ENBASE. The SWHS is thus preserved by DEVS-SCHEME, saved in ENBASE and is retrievable under request later. Figure 6 shows the diagram of the generated system entity structure for SWHS. The command (PE E:SWHS) can display on the screen the generated system entity structure whose name is E:SWHS.

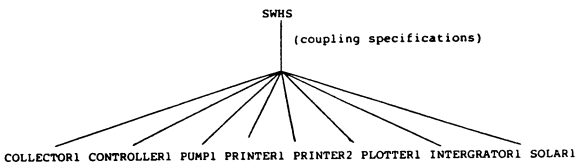

Figure 6. Generated System Entity Structure for SWHS

ACTION 8. Generate Discrete Event Models for TRNSYS Systems

All executable TRNSYS models are coded in FORTRAN and therefore not executable in DEVS-SCHEME. For the management purposes, a command is provided to generate discrete event models for the components of a TRNSYS system. Such models have the same input and output ports as the TRNSYS originals but have only very simple dynamics. These models are stored in the model base (MBASE) and can be reused later (to be discussed in the sequel). The command for this purpose is (SAVE_SYSTEM SWHS).

ACTION 9. Generate TRNSYS_DECK Using SES

The facilities of DEVS-SCHEME enable users to generate TRNSYS_DECK files using functions related to system entity structure rather than going through the above actions repeatedly. Figure 6 can be generated by pruning the system entity structure shown in Figure 1. The command (TRANSFORM E:SWHS) works on the just pruned system entity structure and restores the corresponding system components (e.g., PRINTER1, etc.) in the model base (MBASE). The system SWHS is thus recovered into DEVS-SCHEME environment, therefore the TRNSYS_DECK for SWHS can be again generated (as in Action 6) for another simulation run in TRNSYS.

ACTION 10. Pseudo Simulation

Since the discrete event models in DEVS- SCHEME environment are available, a "pseudo" simulation of SWHS can be performed. The command for this action is (RESTART R:SWHS) where R:SWHS is the simulation starter in DEVS-SCHEME environment. Detailed discussion on how to perform simulation DEVS-SCHEME can be found in (Zeigler 1987). The pseudo simulation is very useful when high level control is needed for the system to be observed and simulated. Research on generating good DEVS versions of continuous systems is in progress in AI and Simulation Group of ECE department at the University of Arizona.

5. Summary

Many more functions are available to manage continuous models by employing system entity structure concepts and facilities in DEVS-SCHEME environment. Current work along this line shows that very little modification of the TRNSYS interface is needed to work with other continuous simulation system, DESIRE, for example. Further investigations into this issue are continuing.

System entity structure management of continuous simulation models in DEVS-SCHEME shows that an interface between discrete and continuous simulation systems can be realized such that the advantages of both can be well utilized and their drawbacks overcome to some extent.

REFERENCE

Kim, T.G., Zeigler, B.P (1987), "The DEVS Formalism : Hierarchical, Modular System Specification in an Object-Oriented Framework", Proc. of 1987 WSC, Atlanta, Georgia.

Rozenblit, J.W., (1985), A Conceptual Bases for Knowledge Based System Design, Doctoral Dissertation, Wayne State University, Detroit.

Rozenblit, J.W., Zeigler, B.P., (1985) "Concepts for Knowledge Based System Design Environment", Proc. Winter Simulation Conference, San Francisco.

TRNSYS Manual, Solar Energy Laboratory, University of Wisconsin at Madison, Dec., 1983.

Zeigler, B.P., (1976) Theory of Modelling and Simulation, John Wiley & Sons.

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

Zeigler, B.P., (1986) "Knowledge Representation from Newton to Minsky and Beyond", Applied Artificial Intelligence.

Zeigler, B.P., (1987) "Hierarchical, Modular Discrete Event Modelling in an Object-oriented Environment", Simulation J., November.

Zeigler, B.P., Zhang, G. (1988) " The System Entity Structure : Knowledge Representation for Simulation Modelling and Design", Artificial Intelligence, Modelling and Simulation, edited by L.E. Widman, et al, John & Wiley & Sons.

Authors' Biographies

Tag Gon Kim is a Research Engineer in Environmental Research Lab at the University of Arizona, where his current responsibility is to design and implement a real time control expert system. From 1980 to 1983, Kim has been a faculty in Department of Electronics and Telecommunication Engineering at National Fisheries University of Pusan, Pusan, Korea. His research interests include knowledge-based simulation and system development, application of AI technology to real-time control expert systems and performance modelling and simulation of parallel computers. He is a member of IEEE/CS, ACM and SCS.

Tag Gon Kim, Research Engineer,
ERLAB, University of Arizona
2601 E. Airport Dr.
Tucson, AZ 85706

Guoqing Zhang is a PH.D. student in the department of Electrical and Computer Engineering at the University of Arizona. In 1984, he got his BSCS at Shanghai Jiao Tong University, Shanghai, China, and in 1987, he finished his MSEE at the University of Arizona. His research interests include Knowledge-based system design methodology, knowledge representation formalism, software system development, and distributed modelling and simulation. Zhang is a member of IEEE/CS and SCS.

Guoqing Zhang
Dept. of Electr. and Computer Engineering
University of Arizona
Tucson, AZ 85721

Bernard P. Zeigler is a professor of Electrical and Computer Engineering 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 Engineering
University of Arizona
Tucson, AZ 85721