Proceedings of the 1993 Winter Simulation Conference
G.W. Evans, M. Mollaghasemi, E.C. Russell, W.E. Biles (eds.)

OBJECT-ORIENTED MEMORY MANAGEMENT IN DEVSIM++

Young C. Kim, Kyung S. Ham, and Tag G. Kim

Department of Electrical Engineering
Korea Advanced Institute of Science and Technology
373-1 Kusong-Dong, Yusong-Ku, Taejon 305-701, KOREA

ABSTRACT

DEVSIM++ is an object-oriented simulation environment which implements Zeigler's modular, hierarchical DEVS formalism and associated abstract simulator concepts in C++. Due to the object-oriented modeling, size of simulation models in DEVSIM++ was bounded to memory size. This paper deals with implementation of an object-oriented memory management scheme as an extension of DEVSIM++. The scheme is based on concepts of persistence data in database management systems. Effectiveness and performance for the extended DEVSIM++ environment will be examined.

1 INTRODUCTION

DEVSIM++ is an object-oriented simulation environment which implements Zeigler's modular, hierarchical DEVS formalism and associated abstract simulator concepts in C++ (Kim and Park, 1992). Since objects are alive during simulation, size of simulation models is bounded to memory size. Our experience of modeling simulation in DEVSIM++ showed that such size limitation is a major obstacle to perform large-scale simulation. To overcome the limitation, a memory management scheme should be developed.

This paper deals with design and implementation of an object-oriented memory management scheme in DEVSIM++. The scheme is based on concepts of persistence data in database management systems. We develop a persistent, object-oriented environment as an extension of C++. Such environment supports program-level data persistence which is transparent to users. Specifically, the environment maintains persistence of objects created in DEVSIM++ while leaving classes defined in DEVSIM++ in memory. For rapid prototyping, we decided to use the EXODUS Storage Manager (Exodus Project 1991, 1992) to store and manipulate persistent DEVSIM++ objects. The storage system manages the allocation of storage for persistent objects both on disk and in memory. It also controls the transfer of objects between disk and memory.

The paper is organized as follows. In section 2 we introduce DEVS formalism and DEVSIM++. Section 3 describes an object-oriented memory management scheme and persistency of models. Finally in section 4 we summarize implementation and experiments of the extended DEVSIM++.

2 DEVS FORMALISM AND DEVSIM++

A set-theoretic formalism, the DEVS formalism (Zeigler 1984, Concepcion 1988) provides the semantics that specifies discrete event models in a hierarchical, modular form. With the semantics, one must specify 1) the basic models, from which larger ones are built, and 2) how these models are connected together in hierarchical fashion. A basic model, called an atomic model (or atomic DEVS), has specifications for the dynamics of the model. An atomic model M is specified as:

$$M = < X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta >$$

X : input events set;
S : sequential states set;
Y : output events set;
δ_{int} : $S \rightarrow S$: internal transition function;
δ_{ext} : $Q \times X \rightarrow S$: external transition function;
$Q = \{(s, e) \mid s \in S, 0 \leq e \leq ta(s)\}$:
total state of M;
$\lambda : S \rightarrow Y$: output function;
$ta : S \rightarrow Real$: time advance function.

The second form of the model, called a coupled model (or coupled DEVS), tells how to couple (connect) several component models together to form a new model. This latter model can be employed as a

component in a larger coupled model, thus giving rise to the construction of complex models in a hierarchical fashion. A coupled model DN is defined as:

$$DN =< D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, SELECT >$$

D : component names set;
for each i in D,
    M_i : DEVS for component i in D;
    I_i : set of influencees of i;
for each j in I_i,
    $Z_{i,j}$: $Y_i \rightarrow X_j$:
    i-to-j output translation function;
$SELECT$: subsets of $D \rightarrow D$:
    tie-breaking selector.

Detail descriptions for the definitions of the atomic and coupled models within the DEVS semantics can be found in Zeigler (1984, 1990).

DEVSIM++ is a realization of the DEVS formalism and the associated simulation algorithms in C++. It also supports facilities for input and output data analysis. For modeling, DEVSIM++ provides the modeler with facilities for the specification of atomic models and coupled models within the DEVS framework, called *Atomic_models* and *Coupled_models*, respectively. For atomic models, the modeler needs to specify transition functions, time advance functions, and output functions: Components, coupling schemes, and selection functions must be specified for coupled models. For simulation, DEVSIM++ implements hierarchical scheduling algorithms in the abstract simulators of atomic and coupled models. Concepts and implementation of such simulators can be found in Zeigler (1984).

DEVSIM++ employs the NIH class library (NIHCL) (Gorlen et al. 1990). The class *Object*, the root class of NIHCL, supports general facilities for operations and queries on objects, including class name, class description, class comparison, and others. The universal class in DEVSIM++ is the class *Entities* and is defined as a subclass of the class *Object*. Figure 1 shows the class hierarchy in DEVSIM++, similar to that of DEVS-Scheme (Zeigler 1990, Kim 1990, 1991).

Models and *Processors*, the main subclass of *Entities* in DEVSIM++, provide the basic constructs needed for modeling and simulation. The class *Models* is further specialized into the major classes *Atomic_models* and *Coupled_models* that realize atomic models and coupled models in the DEVS formalism, respectively.

Atomic_models realize the atomic level of the DEVS formalism. For that, four instance variables for

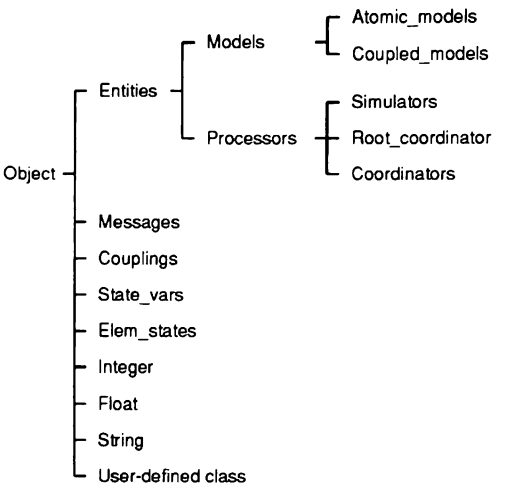


Figure 1: Class Hierarchy in DEVSIM++.

the *Atomic_models* class, *int_transfn*, *ext_transfn*, *outputfn*, and *time_advancefn*, are provided to realize the internal transition function, the external transition function, the output function, and the time advance function of an atomic DEVS model, respectively (Kim 1992). Thus, the development of an atomic DEVS model is to define the four DEVS functions followed by assigning them to the corresponding instance variables.

Coupled_models is the realization of the coupled models definition in the DEVS formalism which embodies the hierarchical construction of modular models. A subclass of *Coupled_models*, *Digraph_models*, specifies a coupled DEVS model in a digraph form. To do so, the class provides facilities for specifying component models and their coupling scheme. Thus, the development of a coupled DEVS model is to specify component models and their coupling scheme using facilities provided by the class. A coupled DEVS model, so developed in a modular form, can be used to construct a yet higher coupled model, resulting in hierarchical construction of modular models.

The class *Processors*, which implements the abstract simulator concepts associated with the DEVS formalism, is a virtual device capable of interpreting models' dynamics. *Processors* has three specializations: *Simulators* for interpreting semantics for *Atomic_models*; *Cordinators* for interpreting semantics for *Coupled_models*; *Root_co_ordinators* for handling overall simulation.

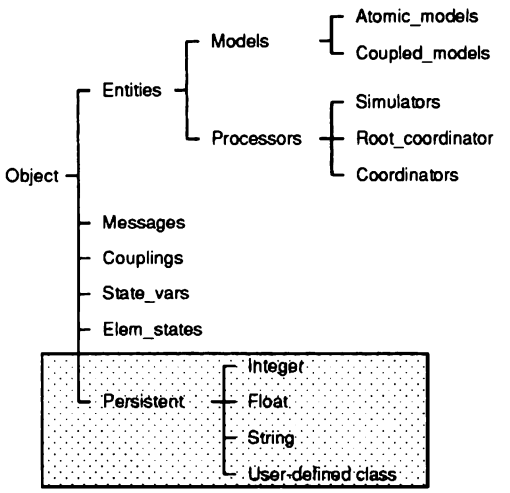


Figure 2: Class Hierarchy in Extended DEVSIM++.

3 MEMORY MANAGEMENT SCHEME

The EXODUS Storage Manager provides basic management support for objects, files, and transactions (Exodus Project 1992, 1993). A storage object is an uninterpreted container of bytes which can range in size from a few bytes to hundreds of megabytes. Storage objects are referenced with object identifiers or OIDs. Objects are allocated in a file, which is simply a collection of related storage objects. Files are used mainly for scans and for clustering related storage objects. Transactions include concurrency control and recovery.

There are "Integer", "Float", "String", and user-defined classes as classes for state variables for DEVS atomic models. To provide the DEVS object-oriented memory management facility, an abstract class "persistent" is defined as a subclass of the root class "Object" in the DEVSIM++ class hierarchy. It would manage almost all aspects of persistence for any derived class such that that's data member is an instance of another class or that has virtual functions. But currently the derived class can not have the following members:

- A pointer to an object of classes for state variables.
- A reference to an object of classes for state variables.

So, we plan to develop the small-sized native storage manager that can be more optimized in performance and size for DEVSIM++ library than the EXODUS Storage Manager. Using such new storage manager, the above problems will be solved.

For simulation, DEVSIM++ implements hierarchical scheduling algorithms in abstract simulators of

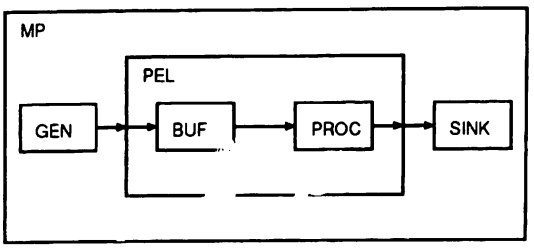
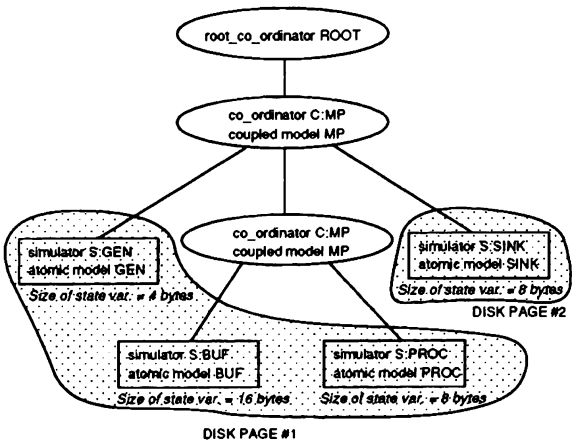


Figure 3: Block Diagram of an Example.



Assumption : The size of one page = 32 bytes

Figure 4: A Simulation Tree and Objects Clustering for the Previous Example.

atomic and coupled models defined by the modelers. In other word, the simulation instances of models are constructed into a tree structure. The leaf nodes of such a simulation tree are instances of atomic models and the internal nodes are coupled models. To overcome the limitation of memory size, all state variables of atomic models are stored in disk as storage objects of the EXODUS Storage Manger. To obtain good performance, it is closely related to how to cluster objects. In current implementation state variables are stored into disk in the depth-first-search oder. This means almost all of atomic models that have the same parent coupled model can be clustered into the same disk page. Figures 3 and 4 shows an example.

Objects needed for simulation are cached into memory by dereferencing through their OIDs. The replacement policy of the unpinned pages is performed using the LRU scheme.

In the side effect of the object-oriented memory management, all states of simulation models are stored as persistent objects if states of coupled models are saved as persistent objects. So, we modified the ab-

stract simulator of DEVSIM++ library to save those of coupled models. After the termination of any simulation due to system failure, user interrupt, or user modeling etc, the simulation can be restarted at the valid state just before the last termination without resimulating from its initial state. This feature can be used for incremental simulation of large-scale system.

4 IMPLEMENTATION AND EXPERIMENTS

Implementation of the memory management scheme described in section 3 is underway. Simulation experiments will be performed for validation and performance measurement for the extended DEVSIM++ environment. First, we will validate persistency for simulation models within the environment. Next, we will measure performance of the environment. Persistency is very useful for large-scale simulation to save the system state from failures and resume the saved state to restart simulation from the point just before such failures occurred. System failure will be simulated by keyboard interrupts and power offs during simulation. When restarting simulation, we will check if the system state before and after such failures be the same. Performance of the extended DEVSIM++ will depend on the ratio of model objects size to memory size. We shall measure overall simulation time for a given size of model objects with the memory buffer size varied. We shall also measure overall simulation time for a given memory buffer size with the model objects size varied.

REFERENCES

A.I. Concepcion and B.P. Zeigler, 1988, "DEVS formalism: A framework for hierarchical model development", *IEEE Trans. on Software Eng.*, Vol. 14, No. 2, pp 228-241.

Exodus Project, 1991, "EXODUS Storage Manager Architectural Overview".

Exodus Project, 1992, "Using the EXODUS Storage Manager V 2.2".

Keith E. Gorlen, Sanford M. Orlow and Perry S. Plexico, 1990, "Data Abstraction and Object-Oriented Programming in C++", John Wiley & Sons, Ltd.

Tag G. Kim and B.P. Zeigler, 1990, "The DEVS-Scheme Simulation and Modelling Environment", Chapter 2 in *Knowledge Based Simulation: Methodology and Application* (eds: Paul A. Fishwick and Richard B. Modjeski) Springer Verlag., Inc., pp. 20-35.

Tag G. Kim, 1991, "Hierarchical Development of Model Classes in The DEVS-Scheme Simulation Environment", *Expert Systems with Applications*, Vol. 3, No. 3, pp. 343-351.

Tag G. Kim and Sung B. Park, 1992, "The DEVS Formalism: Hierarchical Modular Systems Specification in C++", *Proc. of the 1992 European Simulation Multiconference*, pp. 152-156.

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

B.P. Zeigler, 1990, *Object-Oriented Simulation With Hierarchical, Modular Models : Intelligent Agents and Endomorphic Systems* Academic Press, Inc.