DISTRIBUTED OPTIMISTIC SIMULATION OF HIERARCHICAL DEVS MODELS

Ki Hyung Kim
Yeong Rak Seong
Tag Gon Kim
Kyu Ho Park

Department of Electrical Engineering
Korea Advanced Institute of Science and Technology
373-1 Kusong-Dong Yusong-Ku, Taejon 305-701, KOREA
(e-mail: khkim@coregate.kaist.ac.kr)

ABSTRACT

This paper presents a new distributed simulation methodology for hierarchical Discrete Event System Specification(DEVS) models. The DEVS formalism provides a formal basis for specifying discrete event models in a modular, hierarchical form. For distributed simulation of DEVS models, a synchronization mechanism is required to control the advance of simulation clocks. The proposed methodology employs the Time Warp mechanism for such synchronization. However, the Time Warp mechanism must be modified because DEVS and Time Warp have different simulation semantics. This paper deals with such issues. The performance of the proposed methodology is evaluated through a benchmark simulation. Several performance results are presented. The results show that significant speedup can be obtained.

Keywords: Distributed/parallel simulation, DEVS formalism, Time Warp, Object-oriented modeling and simulation

1 INTRODUCTION

The Discrete Event Systems Specification (DEVS) formalism, developed by Zeigler(Zeigler 1984), specifies discrete event models in a hierarchical, modular form. This hierarchical modeling offers such advantages as fast model development, model reuse, and easy model verification and validation(Sargent 1993). These advantages are very helpful in modeling and simulation of large, complex systems.

Parallel simulation of DEVS models differs from traditional logical process-based parallel simulation(Fujimoto 90) in that: (i) the formalism differentiates external and internal events of the models, and (ii) as a simulation mechanism, the DEVS abstract simulators are used(Zeigler 1984). Owing to these differences, most of parallel DEVS approaches exploit the specific parallelism of the DEVS formalism. Such approaches can be broadly classified into two approaches: *synchronous* and *asynchronous* ones. Synchronous approaches use a unique global scheduler to synchronize the simulation progress across all of the computer nodes. That is, only events with the same simulation time are executed in parallel, and thus, the global scheduler becomes a bottleneck for speedup. Asynchronous approaches allow each computer node to have different simulation clock. Thus, the bottleneck can be greatly reduced. As an asynchronous approach, DEVS-Ada/TW(Christensen and Zeigler 1990) employed the Time Warp mechanism for global synchronization. However, it treats all models mapped in one computer node as one logical process. Thus, the size of a logical process becomes larger, and the rollback cost increases. Moreover, in DEVS-Ada/TW, a hierarchical DEVS model must be partitioned only at the top level of its hierarchy. Thus, The mapping of a hierarchical DEVS model onto parallel computers becomes difficult.

This paper proposes a new distributed simulation methodology, called the Distributed Optimistic Hierarchical Simulation (DOHS) scheme, for hierarchical DEVS models. The DOHS scheme is based on the DEVS abstract simulators and the Time Warp mechanism. The combining of DEVS and Time Warp causes some problems because DEVS and Time Warp have different simulation semantics. We also refer such issues in this paper.

The proposed methodology has several advantages compared to the previous approaches. First, it can utilize both the synchronous and asynchronous parallelism of DEVS models. Second, it supports flexible partitioning and mapping of hierarchical DEVS models. Finally, it's rollback algorithm is more efficient than the previous approaches because each DEVS model can roll back individually.

The rest of the paper is organized as follows: Section 2 describes an overview of the DEVS formalism and its abstract simulator. Section 3 presents the proposed DOHS scheme, including the combining issues of DEVS and Time Warp. Section 4 presents the realization of the DOHS scheme. In Section 5, the performance of the

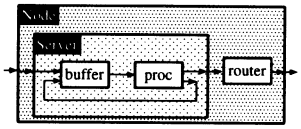
DOHS scheme is measured through benchmark simulation. Section 6 draws the conclusion.

2 DEVS FORMALISM AND ABSTRACT SIMULATORS

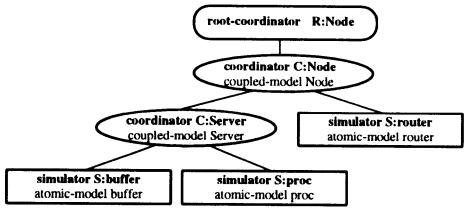
The DEVS formalism specifies discrete event models in a hierarchical, modular form(Zeigler 1984). To hierarchically construct DEVS models, the formalism specifies two types of models. One is atomic model and another is coupled model. An atomic model specifies the behavior of the model by external and internal transition functions. The external transition function specifies the action for external inputs from other models, and the internal transition function specifies the action without external inputs. A coupled model specifies how to couple several component models together to form a new model. A coupled model can be employed as a component in a larger coupled model. Thus, complex models can be constructed in a hierarchical, modular form. As an example, Figure 1 (a) shows a simple queuing node model, *Node*. *Node* consists of coupled model *Server* and atomic model *router*; *Server* is composed of two atomic models, *buffer* and *proc*. Detailed descriptions for the definitions of the atomic and coupled DEVS can be found in (Zeigler 1984).

To simulate a DEVS model, the abstract simulator was developed(Zeigler 1984). The abstract simulator interprets the behavior of its associated DEVS model. There are two kinds of abstract simulators, *simulator* for atomic models and *coordinator* for coupled models. The *simulators* and *coordinators* are linked by the coupling information of the corresponding coupled models, thus forming the same hierarchical structure as that of the models. As an example, Figure 1 (b) shows the hierarchical abstract simulators of model *Node*. The simulation progress is managed by the global scheduler, named *root-coordinator*.

An abstract simulator communicates with the external world by explicitly typed messages, *i.e.* $(*)$, (x), (y), and $(done)$. A $(*)$ message indicates when an internal transition function needs to be activated. When a *simulator* receives a $(*)$ message, it generates output messages, (y)'s, by the output function of its associated model, and executes the internal transition function. Note that in the DEVS formalism, output messages can be produced only while executing internal events. An (x) message indicates that an external event x from other *simulator* is arriving. Thus, when receiving an (x) message, a *simulator* executes the external event transition function of its associated model. Finally, after the execution of a $(*)$ or (x) message, a *simulator* (or *coordinator*) notifies the parent *coordinator* of its next scheduled time, t_N, by a $(done)$ message.



(a) A queuing node model, Node



(b) Abstract simulators of model Node

Figure 1: A hierarchical queuing node model and its associated abstract simulator

3 DISTRIBUTED OPTIMISTIC HIERARCHICAL SIMULATION OF DEVS MODELS

This section presents the proposed distributed simulation methodology, the Distributed Optimistic Hierarchical Simulation (DOHS) scheme. This methodology is based on the existing hierarchical simulation mechanism given in the previous section and the Time Warp mechanism. The hierarchical simulation mechanism should be modified to fit into the parallel simulation environment. A Time Warp algorithm is devised to be incorporated into the hierarchical simulation mechanism.

Before describing the DOHS scheme, we first discuss the problems occurring from the semantic differences of Time Warp and DEVS.

3.1 Combining Issues of Time Warp and DEVS

Basically, the simulation time advance mechanism of DEVS differs from that of Time Warp. In Time Warp, the simulation time advances during a message transmission, not in a logical process. Thus, a message has send and receive time-stamps for advancing the simulation time. However, in the DEVS semantics, the simulation time advances in an abstract simulator (or its associated model), not during a message transmission. Thus, a message has only one time-stamp, and instead, an abstract simulator holds an elapsed-time-stamp which holds the time elapsed since the last event. On receiving an external event message, an abstract simulator emits output event messages when the elapsed-time advances to $ta(s)$, the time advance value specified by the formalism. Figure 2 depicts this difference.

The causality error is a kind of error that the future affects the past. The requirement not to commit such errors is called the global causality constraint. For satisfying this

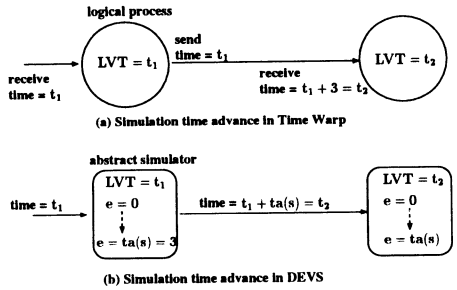


Figure 2: Simulation time advance mechanism

constraint, the Time Warp mechanism demands the following two semantic rules:

Rule 1. For each logical process, each incoming message has a distinct receive time-stamp.

Rule 2. The receive time-stamp of a message must be larger than its send time-stamp.

To employ the Time Warp mechanism in the DEVS simulation, the above two rules must be satisfied. However, the DEVS formalism does not preserve the rules because the formalism is a general discrete event system specification. DEVS admits that multiple external events with the same simulation time may arrive to one influencee *simulator*, as shown in Figure 3. This leads to the disagreement of the above *Rule 1*, since (x_{AC}, t) and (x_{BC}, t) have the same time-stamp, t.

Furthermore, the DEVS formalism does not guarantee the *Rule 2*. The formalism allows that the time advance function, $ta(s)$, may return zero value. As an example, let's consider the case that $ta(s) = 0$ for some state s, as shown in Figure 4. There are three *simulators*, A, B, and C. At $t2$, *simulator B* generates external event messages, $(x_{BA}, t2)$ and $(x_{BC}, t2)$; the messages are transmitted to *simulator A* and C respectively. After executing the received message $(x_{BA}, t2)$, *simulator A* sees that $ta(s) = 0$ and thus, immediately issues external event message $(x_{AC}, t2)$ to *simulator C*. Therefore, *simulator C* receives two external event messages, $(x_{BC}, t2)$ and $(x_{AC}, t2)$, having the same time-stamp $t2$. Even if the two messages have the same time-stamp, $(x_{BC}, t2)$ must be executed earlier than $(x_{AC}, t2)$. This is a *causality relation*. However, *simulator C* cannot know which one is to be processed first; thus, a *causality error* occurs.

To satisfy the above two semantic rules, the *time-and-priority-stamp* and the *causality requirement* are devised in the DOHS scheme. First, for rule (a), all (x) and $(*)$ messages are assigned a *time-and-priority-stamp*. For a *simulator*, input (x) and $(*)$ messages are ordered by the lexicographic ordering of their *time-and-priority-stamps*, as shown in the following definition.

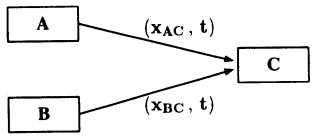


Figure 3: A conflict in input message ordering

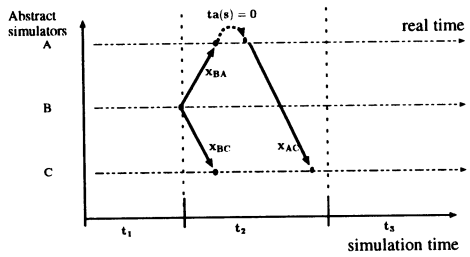


Figure 4: An example of a causality relation

Definition 1 Let both m_1 and m_2 be either (x) or $(*)$ messages, and let $tp_1 =< t_1, p_1 >$ and $tp_2 =< t_2, p_2 >$ be the ***time-and-priority-stamps*** of m_1 and m_2 respectively. Then, $tp_1 < tp_2$, if and only if *(i)* $t_1 < t_2$, or *(ii)* $t_1 = t_2$ and $p_1 < p_2$.

By using the *time-and-priority-stamp*, all (x) and $(*)$ messages in a *simulator* can be strictly ordered.

The priority in the *time-and-priority–stamp* can be obtained from the DEVS formalism's *select* function. The *select* function gives a unique priority to each atomic DEVS model (or its associated *simulator*). That is, the priority-stamp of an (x) message is the priority of the message's source *simulator*. The $(*)$ message's priority-stamp is smaller than any of the received (x) messages in a receiving *simulator*. This is because the internal event transition must occur before any external event transition with the same simulation time. Thus, all input (x) and $(*)$ messages can get a distinct *time-and-priority* stamp in a receiving *simulator*. As an example, let's consider the case in Figure 3. Let's assume that *simulator A* has smaller priority than *simulator B*. Then, (x_{AC}, tp_1) will be processed first than (x_{BC}, tp_2), because $tp_1 < tp_2$

Second, for semantic rule (b), we assert the ***causality requirement***. The *causality requirement* asserts that the time advance function, $ta(s)$, must result in a positive value for all states:

$$ta(s) > 0, \text{ for all states } s.$$

The *causality requirement* can prevent the causality errors.

However, the causality requirement might restrict the modeling flexibility of the DEVS formalism. Also, this makes sequential DEVS models to be translated for parallel simulation, because DEVS models can have zero time

advance value in the sequential simulation. The DOHS scheme solves this problem by substituting ε for a zero time advance value, where ε is a minimum positive time advance value, such that:

$\varepsilon * A < K$, for any positive constant A,

where K is the minimum of any explicitly specified positive time advance value for all models. That is, in the modeling perspective, models can have zero time advance value; during a simulation, the zero value is substituted by ε, thus satisfying the *causality requirement*. By employing this mechanism, all sequential DEVS models can be simulated in the DOHS scheme. As an example, let's consider the case in Figure 4. The zero time advance value of *simulator A* will be substituted by ε. Thus, $(x_{AC}, tp2)$ will have larger *time-and-priority-stamp* than $(x_{BC}, tp1)$ by ε.

3.2 Overview of DOHS scheme

Figure 5 shows the global structure of the DOHS scheme. In each computer node, the DOHS scheme consists of *node-coordinator*, distributed abstract simulators, *DOHS-manager*, and *DOHS-queue*. The *node-coordinator* schedules the distributed abstract simulators. The *DOHS-manager* and *DOHS-queue* manage the global synchronization by the Time Warp mechanism. They handle all messages from/to the distributed abstract simulators. The distributed abstract simulators perform actual simulation for DEVS models.

The *node-coordinator* is a distributed version of the global *root-coordinator*. For asynchronous distributed simulation, each computer node must schedule by itself. That is, the global scheduler, *root-coordinator*, must be distributed to each node. In each computer node, a *node-coordinator* schedules the abstract simulators mapped in the node.

Since each computer node can have different local clock, messages with different time-stamps arrive from other computer nodes. Thus, to manage them, the *DOHS-manager* and *DOHS-queue* are developed. They perform three functions. First, they manage simulation progress by handling all messages from/to the distributed abstract simulators. The *DOHS-queue* is basically a priority queue, in which all messages are inserted by their time-stamps order. The *DOHS-manager* fetches the first message from the *DOHS-queue* and sends it to its destination abstract simulator. Second, when a straggler message arrives, they find it and initiate rollbacks. Finally, they perform the global control mechanism, which will be described later.

The distributed abstract simulators perform actual simulation. In the DOHS scheme, the abstract simulator algorithms must be modified to support a rollback. For this, parallel versions of *simulator* and *coordinator* are developed, named as *p-simulator* and *p-coordinator* respectively. When a straggler message arrives, a *p-simulator*

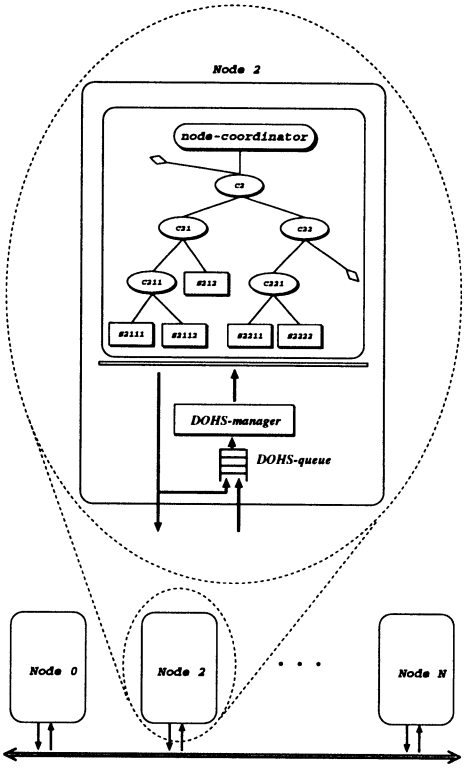


Figure 5: The Overview of the DOHS scheme

rolls back to the previous state and recomputes the reordered messages. After a rollback, the *p-simulator* is rescheduled hierarchically by its parent *p-coordinators*.

The DOHS scheme can be divided into two major parts: the local control mechanism and the global control mechanism. The local control mechanism ensures that messages are eventually executed in correct order. The mechanism allows out-of-order execution of messages; such causality errors are corrected by using rollbacks and recomputations. The global control mechanism is concerned with global issues, such as I/O handling, termination detection, memory management, and flow control.

3.3 Local Control Mechanism

The local control mechanism uses rollbacks and recomputations for recovering the causality errors. For a rollback, each *p-simulator* manages three data structures: an *input history*, a *state history*, and an *output history*. The input history stores incoming (x) or $(*)$ messages in increasing *time-and-priority-stamp* order. The state history saves state snapshots whenever an input message is processed. The output history contains the inverse image of the output messages sent.

The rollback operation of the DOHS scheme is more complex than that of the traditional Time Warp mecha-

nism. This is because *p-simulators* are scheduled hierarchically by several *p-coordinators*. Thus, scheduling informations are distributed across several *p-coordinators*.

The rollback operation consists of three steps. First, current hierarchical schedules must be cancelled, since the straggler message will make a new schedule. For such cancelling. a schedule cancelation algorithm is developed. The algorithms cancels the scheduling informations in several *p-coordinators*. After the schedule cancelation, the *DOHS-manager* transmits the straggler message to the affected *p-simulators*. The second step of the rollback occurs in the affected *p-simulators*. The *p-simulators* roll back their state and recompute the reordered messages by the developed rollback algorithm. Finally, after this re-computation, the *p-simulators* are re-scheduled by their parent *p-coordinators*, since their local clocks were changed.

3.4 Global Control Mechanism

The global control mechanism is performed by the *DOHS-manager*. The basic algorithm for the GVT calculation is the same as Time Warp, except the definition of GVT which is modified as follows.

Definition 2 Let *min(DOHS-queue)* be the minimum time-stamp value of all messages in *DOHS-queue*. Then, the ***Local Virtual Time (LVT)*** of a computer node is defined as the lowest value between *min(DOHS-queue)* and the *last event time*, t_L, of the *node-coordinator*. The ***Global Virtual Time (GVT)*** is the minimum of *LVT*'s of all computer nodes.

After the calculation of GVT, *DOHS-manager* collects fossils in all *simulators*, such as useless memories. When GVT becomes infinity, *DOHS-manager* terminates the simulation.

4 REALIZATION OF DOHS SCHEME

The DOHS scheme developed in the previous section serves as a distributed simulation methodology of DEVS models. It remains to show how the DOHS scheme can be implemented on the parallel machine. The DOHS scheme is realized in D-DEVSim++ by C++. D-DEVSim++ is a parallel extension of DEVSim++, a sequential DEVS modeling and simulation environment(Kim and Park 1992). Thus, D-DEVSim++ inherits all features of DEVSim++. D-DEVSim++ is implemented on KAICUBE860, which is a 5-dimensional hypercube parallel machine developed at KAIST. Each computer node contains a 40 MHz i860 microprocessor and 5 communication channels which employ the store-and-forward routing scheme.

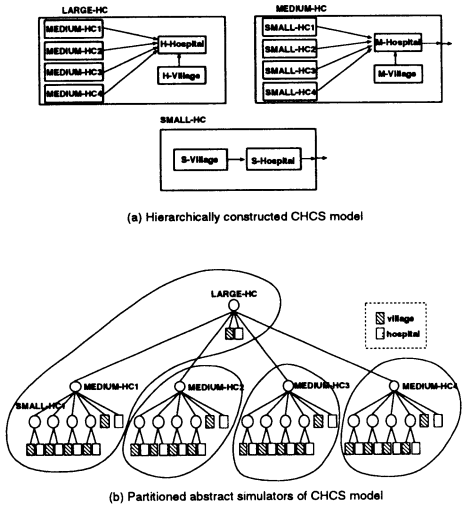


Figure 6: The hierarchical model of the Columbian health care system

5 EXPERIMENTAL RESULTS

In this section, the proposed DOHS scheme is evaluated by experiments. For performance evaluation, the Columbian Health Care System(CHCS) queuing benchmark model is used. This model has been used to analyze the performance of various researches(Baezner *et al.* 1988). The model is one of a multi-tiered health care system of villages and health centers. There is one health center for each village. When villagers become ill, they travel to their local health center for assessment and are treated when possible. If they cannot be treated locally, patients are referred up the hierarchy to a next health center, where the assessment/treatment/referral process is repeated. Upon arriving at a health center, a patient is enqueued until one of health care worker becomes available. It is assumed that patients can always be treated at the top level hospital of the health care system.

Figure 6 (a) depicts the hierarchically constructed CHCS model. The model is designed by the hierarchical modeling methodology. The highest level coupled model is LARGE-HC. LARGE-HC is decomposed into 4 MEDIUM-HC' which are also decomposed into 4 SMALL-HC'. Each HC has two atomic models: Hospital and Village. As an experimental frame, each village is a source which generates a fixed number of patients. Each hospital becomes a server which contains a waiting queue and a set of doctors. It is assumed that each first/second/third-level hospital has 16/4/1 doctors respectively. For parallel simulation, the CHCS model is partitioned. Figure 6 (b) shows one possible partition for 4 computer nodes.

For performance evaluation of the DOHS scheme, the

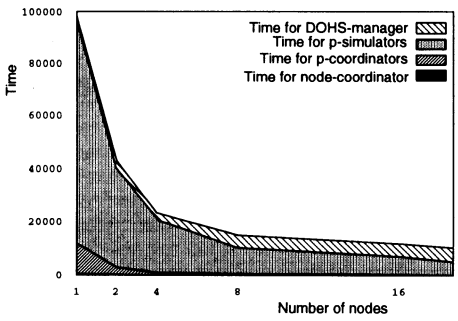


Figure 7: Simulation time vs. number of nodes

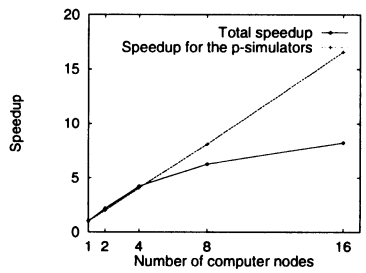


Figure 8: Speedups vs. number of nodes

simulation time for the CHCS model is measured while the number of computer nodes is varied from 1 to 16. Note that when the number of computer nodes is 1, D-DEVSim++ runs in the sequential mode. In the sequential mode, D-DEVSim++ performs no operation related to global synchronization such as, *GVT* calculation, fossil collection, state saving, and rollback. Experimental results are shown in Figure 7. In addition to the total simulation time, the simulation time taken by each part of the DOHS scheme is also measured: that is, the time taken by *p-simulators*, *p-coordinators*, *node-coordinator*, and *DOHS-manager*. The corresponding speedups of the results are shown in Figure 8.

The slope of the total speedup curve declines at 8 and 16 nodes. This is mostly due to unbalanced load distribution. Most early ended computer nodes wait until the simulation in the latest computer node is done. Such waiting time is included in the *DOHS-manager* curve of Figure 7. Since we partitioned the CHCS model intuitively, this waiting time becomes dominant as the number of computer nodes increases. The *DOHS-manager* curve also includes the message handling time of *DOHS-queue*. This time is very small compared to the time spent in the *p-coordinators* and *node-coordinator*, as shown in Figure 7. That is, when the number of computer nodes is 1, this time is just 1.5% of the total simulation time.

The *p-simulators* curve shows very high speedup as shown in Figure 8. Since atomic models express the behavior of a system, this time can be considered as the minimum simulation time. By reducing the waiting time due to the unbalanced load, significant speedup can be obtained.

6 CONCLUSION

This paper proposed a new distributed simulation methodology for hierarchical DEVS models, the Distributed Optimistic Hierarchical Simulation (DOHS) scheme. The DOHS scheme is based on the DEVS abstract simulators and the Time Warp mechanism. DEVS and Time Warp have different simulation semantics. To compensate this differences, the *time-and-priority-stamp* and *causality requirement* were devised. The DOHS scheme was described. To distribute the global scheduler, the *node-coordinator* was developed. To manage the simulation and synchronization in each computer node, the *DOHS-manager* and *DOHS-queue* were developed. Also, to support a rollback, parallel abstract simulator algorithms for atomic and coupled models were devised. The performance of the DOHS scheme was evaluated through the simulation of Columbian health care benchmark models. We confirmed that significant speedup can be obtained by the DOHS scheme.

REFERENCES

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

R. Sargent. "Hierarchical modeling for discrete event simulation (panel)" In *Proceedings of the 1993 Winter Simulation Conference* p569

E.R. Christensen and Bernard P. Zeigler. 1990. "Distributed Discrete Event Simulation: Combining DEVS and Time Warp" In *Proceedings of the SCS Eastern Multiconference on AI and Simulation Theory and Applications*

R.M. Fujimoto. 1990. "Optimistic Approaches to Parallel Discrete Event Simulation" *Trans. of The Society for Computer Simulation* vol. 7, no. 2 (Oct.):153-191.

Kim, T. G., and S. B. Park. 1992. The DEVS Formalism: Hierarchical Modular Systems Specification in C++. In *Proc. of the 1992 European Simulation Multiconference*, 152–156.

D. Baezner, G. Lomow, and B.W. Unger. 1990. "Sim++: The Transition to Distributed Simulation." In *Proceedings of the SCS Multiconference on Distributed Simulation*