DEVS 모델의 낙관적 분산 시뮬레이션을 위한 사건 정렬 방법

Events Ordering in Optimistic Distributed Simulation of DEVS Models

김기형*, 김탁곤*, 박규호*
Ki Hyung Kim, Tag Gon Kim and Kyu Ho Park

Abstract

In this paper, we propose a new events ordering mechanism for the optimistic distributed simulation of DEVS models. To simulate DEVS models in a distributed environment, a synchronization protocol is required for correct simulation. Time Warp is the most well-known optimistic synchronization protocol for distributed simulation. However, employing the Time Warp protocol in distributed simulation of DEVS models incurs events ordering problem due to the semantic difference between Time Warp and DEVS. Thus, to resolve such semantic difference, we devise the time-and-priority-stamp and ε-delay schemes. The proposed schemes can order simultaneous events correctly in Time Warp-based distributed simulation of DEVS models.

Keywords : Parallel processing, Distributed/parallel simulation, DEVS formalism, Time Warp, Simultaneous events

1. Introduction

Discrete event simulation is frequently needed in analyzing and predicting performance of systems. However, simulation of large, complex systems remains a major stumbling block due to its prohibitive computational costs. Distributed discrete event simulation (or shortly, distributed simulation) offers one approach that can significantly reduce these computational costs. Since distributed simulation deals with large and complex systems, model verification and validation become important problems. The DEVS (Discrete Event Systems Specification) formalism, developed by Zeigler[20], provides a formal framework for specifying discrete event models in a hierarchical, modular manner. This hierarchical modeling capability offers such advantages as fast model development, model reuse, and easy model verification and validation[17]. Thus, the DEVS formalism can ease the model verification and validation problems of distributed simulation.

In distributed simulation, one of the most difficult problem is to handle the *causality error* which is a kind of errors

* Department of Electrical Engineering, Korea Advanced Institutes of Science and Technology.

that the processing of future events affects the processing of past events. To prevent such causality errors, a synchronization algorithm is required. Many synchronization algorithms have been proposed. Such algorithms can be classified into two classes: *synchronous* and *asynchronous* ones. Synchronous algorithms use a central event scheduler to synchronize the simulation progress across all of the processors which are involved in the simulation. Since the simulation time is managed by the central scheduler, only the events with the same simulation time can be parallelized. Asynchronous algorithms allow each processor to have different simulation clock. For this, asynchronous algorithms rely on the distributed synchronization protocols for synchronization instead of the central scheduler. In asynchronous algorithms, two classes of protocols are mainly employed: *conservative* one[16], which always prevent the causality error, and optimistic one[6], which can detect and resolve the conflict of causality.

Time Warp is the most well-known optimistic synchronization protocol for distributed simulation [6]. By employing Time Warp in the distributed simulation of DEVS models, we can achieve a significant speedup of the simulation time [8, 7]. Time Warp takes an optimistic approach to resolve the conflict of causality. That is, Time Warp permits causality errors to occur; however, when a causality error is detected, a rollback occurs to repair it. For detecting a causality error, Time Warp assumes that all input events of a model can be ordered totally[6]. However, the DEVS formalism has its own semantics since it has been motivated from system theory; thus, it does not satisfy this assumption of Time Warp. For example, in the distributed simulation of DEVS models, *simultaneous events* whch have the same time-stamp can occur. Such simultaneous events cannot be ordered. Thus, to adjust the semantic difference between Time Warp and the DEVS formalism, we should develop an events ordering mechanism which makes DEVS models meet this assumption.

There have been several researches about the events ordering problem in the distributed simulation of DEVS models. The Extended DEVS (E-DEVS) formalism[19] proposed a new order function for ordering the simultaneous events of certain DEVS models in distributed DEVS simulation. P-DEVSim++[18] solved this simultaneous events ordering in distributed DEVS simulation by introducing the concept of priority. However, these approaches are based on the synchronous simulation algorithms and cannot be used for the asynchronous simulation algorithms. Chow [3] proposed a parallel extension of the DEVS formalism, called the parallel DEVS, to handle the parallelism in simultaneous events. Meanwhile, in the traditional distributed simulation domain, the ordering problem of simultaneous events have been researched as follows. Cota and Sargent [5] showed that the dependency order of simultaneous events in distributed simulation can be enforced in sequential simulation by using a particular assignment of priorities to the models. Agre and Tinker[1] employed the *rank* and *msgID* into a time-stamp to assign a globally unique time-stamp to an event. Mehl[15] employed the concept of *age* and *id* into a time-stamp for the same reason. However, their approaches cannot be employed in the distributed DEVS simulation because the simultaneous events of DEVS models occur in more complex ways than that of the traditional distributed simulation.

In this paper, we analyze the characteristics of DEVS events and propose a new events ordering mechanism. The proposed mechanism can make DEVS models meet the model assumption of Time Warp. Also the mechanism can order all DEVS events correctly with respect to their causal relationships.

2. Backgrounds

2.1 DEVS Formalism

The DEVS formalism provides a basis for specifying discrete event models in a hierarchical, modular form[20]. In the DEVS formalism, one must specify the basic models from which larger ones are built and indicate how these models are connected together in a hierarchical fashion. The basic model *AM*, also called the atomic model, is defined by a 7-tuple[20]:

$AM = \langle X, S, Y, \delta_{int}, \delta_{ext}, \lambda, ta \rangle$

where,
X : external input events set
S : sequential states set.
Y : external output events set
$\delta_{int} : S \times \{i\} \rightarrow S$: internal transition function
where i is an internal event which notifies that the next schedule time has arrived.
$\delta_{ext} : Q \times X \rightarrow S$: external transition function,
where Q is the set of the total states of M given by
$Q = \{ (s, e) \mid s \in S \text{ and } 0 \le e \le ta(s) \}$
$\lambda : S \times \{i\} \rightarrow Y$: output function
$ta : S \rightarrow R^{+}_{0,\infty}$: time advane function

As with modular specifications in general, we must view the above atomic DEVS model as possessing input and output ports through which all interactions with the external world are mediated. To be more specific, when external input events are arriving from other model and received on its input ports, the model decides how to respond to them by its external transition function. In addition, when no external events arrive until the next internal transition time which is specified by the time advance function, the model changes its state by the internal transition function and reveals itself as external output events on the output ports to be transmitted to other models. For the notice of the next internal transition time, an internal event *(i)* is used as shown in the above definition.

Several atomic models may be coupled in the DEVS formalism to form a multi-component model, also called a coupled model. In addition, closed under coupling, a coupled model can be represented as an equivalent atomic model. Thus, a coupled model can itself be employed as a component in a larger coupled model, thereby giving rise to the construction of complex models in a hierarchical fashion. Also, the coupled model specifies a tie-breaking rule, called the SELECT function, which is used for ordering simultaneous events in sequential simulation. The use of the SELECT function will be described later. A coupled model *CM* is defined as follows[20]:

$CM = \langle D, \{M_i\}, \{I_i\}, \{Z_{i,j}\}, SELECT \rangle$
where
D : set of component names.
For each i in D
M_i : DEVS for componet i.
I_i : set of influencees of i.
For each j in it I_i
$Z_{ij} : Y_i \rightarrow X_j$: i-to-j output translation function.
$SELECT$: subsets of $D \rightarrow D$: tie-breaking function.

Detailed descriptions for the definitions of the atomic and coupled DEVS can be found in [20, 21].

For simulation of DEVS models, the abstract simulator concept was developed by Zeigler[20]. This abstract simulator concept have been employed in both sequential and distributed simulation environments of DEVS models[13, 12, 4, 18, 19, 11, 10]. An abstract simulator is a virtual processor which interprets the behavior of a DEVS model. For simulation, an abstract simulator is assigned to each DEVS model in a one-to-one fashion; thus, abstract simulators form the same hierarchical structure as that of the models. Abstract simulators simulate DEVS models by exchanging event messages with each other.

2.2 Time Warp protocol

The Time Warp protocol is the most well-known optimistic synchronization protocol for distributed simulation[6]. In Time Warp, the simulated system is partitioned into a set of subsystems that are simulated by a set of logical processes which communicate by sending/receiving events. For scheduling, *send* and *receive* time-stamps are assigned to an event. The receive time-stamp represents the scheduled simulation time of the event, while the send time-stamp represents the simulation time the event was sent. Time Warp takes an optimistic approach for synchronization, in that a process executes every message as soon as it arrives. If a message with a smaller receive time-stamp subsequently arrives (this message is called a straggler), the process rolls back its state to the time-stamp of the straggler and re-executes from that point. To guarantee the detection of a straggler message, all input messages should be ordered totally. For this, Time Warp assumes the following two semantic rules for models [6].

Rule 1. For each simulator, every incoming event has a distinct receive time-stamp.

Rule 2. The send time-stamp of an event must be smaller than its receive time-stamp.

These rules are proved to be sufficient for total ordering of input events in each model[6,14]

3 Ordering of DEVS Events in the Optimistic Distributed Simulation

Distributed simulation can exploit the parallelism of DEVS models. In this paper, we assume the following distributed simulation model based on the Time Warp protocol. Basic simulation mechanism combines abstract simulator-based hierarchical simulation and the Time Warp protocol. That is, for simulation, each atomic DEVS model is assigned to its own simulator. Simulators are distributed across computer nodes; they simulate DEVS models by exchanging time-stamped event messages with each other. Each computer node has its own local simulation clock, local scheduler, and simulators mapped on it. The local scheduler manages corresponding local simulation clock and generates internal events for the simulators waiting for internal transitions. For synchronization between computer nodes, the Time Warp protocol is employed. Refer to [8, 9, 10, 7] for more detailed description of the model.

However, DEVS models don't satisfy the two semantic rules of the Time Warp protocol because the DEVS formalism is a general formalism based on system theory. Thus, to employ the Time Warp protocol in the simulation of DEVS models, we should resolve the semantic differences of Time Warp and DEVS. Simultaneous events are defined as the events scheduled to occur at the same simulation time; thus, they have the same time-stamps. During simulation of DEVS models, a simulator can reeive simultaneous input events; this violates Rule 1. Also, the simulation time advance mechanism of DEVS differs from that of Time Warp. In Time Warp, the simulation time advances during an event transmission, not in a logical process. Thus, an event has *send* and *receive* time-stamps for advancing the simulation time. The receive time-stamp represents the scheduled simulation time of the event, while the send time-stamp represents the simulation time the event was sent. However, in the DEVS semantics, the simulation time advances within simulators (or their associated atomic DEVS models), not during a message transmission. The sending and receiving of an event occurs at the same simulation time; thereby an event needs only one time-stamp. This violates Rule 2. Thus, to adjust this semantic difference between Time Warp and DEVS, we should develop an events ordering mechanism which makes DEVS models meet these rules.

The simultaneous events of DEVS models can be divided into two classes: *internal* and *external* simultaneous events. Internal simultaneous events are generated when more than one simulator have the same next internal transition time; then, the local scheduler issues multiple internal events for the simulators. These internal events are simultaneous events since they would be executed all at the same simulation time. During executing these internal events, simulators may generate multiple external (output) events. These events are external simultaneous events. Figure 1 shows these two classes of simultaneous events. In Figure 1, the horizontal direction represents time - both real and simulation times - and the vertical direction represents simulator (or model) space. At simulation time 9, i_A and i_C are internal events, while x_{AB}, x_{CD}, and x_{CE} are the external events caused by them. Moreover, these two classes of simultaneous events may occur repeatedly at the same simulation time. That is, after executing external simultaneous events, the affected simulators may want to execute their internal transitions at the same simulation time, since the range of the time advance function includes zero. Thus, local schedulers generate another set of subsequent internal events for the simulators, thereby repeating the entire process. For example, in Figure 1, i_B and i_E are the subsequent internal simultaneous events caused by the external events, x_{AB} and x_{CE}. Therefore, all of

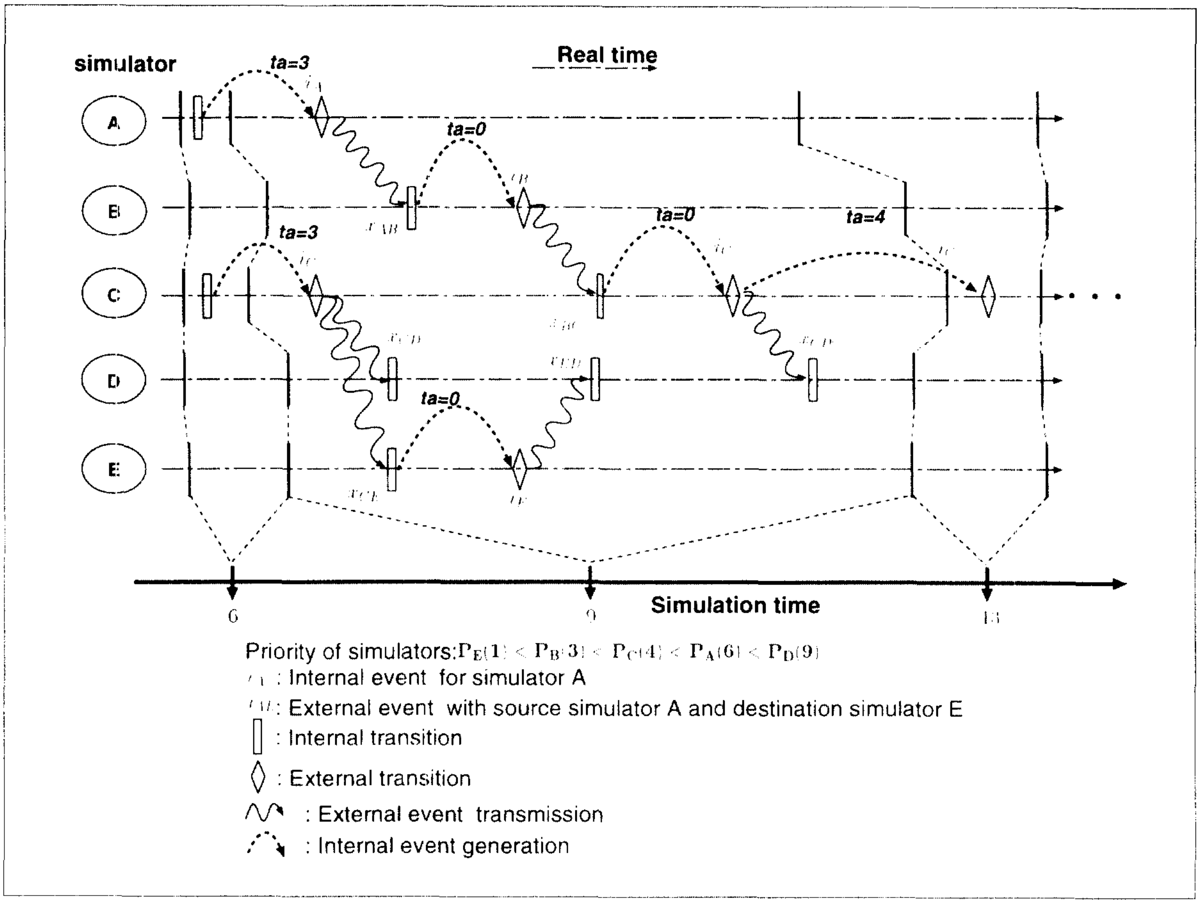
simulator
Real time
ta=3
ta=0
ta=3
ta=0
ta=4
ta=0
A
B
C
D
E
Simulation time
6
9
13
Priority of simulators: P_E(1) < P_B(3) < P_C(4) < P_A(6) < P_D(9)
: Internal event for simulator A
: External event with source simulator A and destination simulator E
: Internal transition
: External transition
: External event transmission
: Internal event generation

Figure 1 : Simultaneous events of DEVS models

these simultaneous events should be simulated in the correct order.

To order these simultaneous events correctly, we should preserve their causal relationships. The DEVS simultaneous events have two kinds of causal relationships: *direct* and *indirect*. A causal relationship between the causing internal event and the resulting external event is called direct since the execution of an internal event can generate an external event directly in DEVS semantics, as mentioned before. A causal relationship between causing and resulting external events is called indirect since an external event can be only generated by an internal event, not by an external event. This indirect causal relationship occurs because the range of the time advance function of a DEVS model includes zero. Thus, after executing the causing external event, a simulator can receive an internal event at the same simulation time; the execution of this internal event generates the resulting external event. For example, in Figure 1, the solid arrow between events i_B and x_{BC} represents their direct causal relationship. Also, the dotted and solid arrows between events x_{AB} and x_{BC} represents their indirect causal relationship.

For ordering these simultaneous events while preserving their direct and indirect causal relationships, we propose the *time-and-priority-stamp* and ε-delay schemes. First, the *time-and-priority-stamp* scheme is developed as a new ordering mechanism for external and internal events. The time-and-priority-stamp consists of a pair, $\langle t, p\rangle$, where t is the simulation time and p is the priority. Note that the lower value p implies the higher priority (if we assume only non-negative priorities, the priority-stamp value 0 has the highest priority). That is, since simultaneous events have the same time-stamps, we add a priority-stamp for ordering such events. These pairs are ordered lexicographically as shown in the following definition.

definition 1 *Let m_1 and m_2 be either internal or external events , and let $tp_1=\langle t_1, p_1\rangle$ and $tp_2=\langle t_2, p_2\rangle$ be the time-and-priority-stamp 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 $p_1 < p_2)$.*

The rules for assigning the time-and-priority-stamp are different for internal and external events. The time-stamp of an internal event is its scheduled simulation time. The time-stamp of an external event is the same as that of its causing internal event (note that external events can be generated during the execution of an internal event). The priority-stamp utilizes the SELECT function, the tie-breaking function of the DEVS formalism. The SELECT function assigns a unique priority to each simulator (or its associated atomic DEVS model) before the simulation starts. The priority-stamp of an external event is the priority of its source simulator-the simulator which sends the event. The priority-stamp of an internal event is higher (or smaller) than those of any external events. For example, Figure 2 shows the assignment of the time-and-priority-stamps to the simultaneous events of our previous example. The priority-stamp of x_{AB} *is* 6, the priority of simulator *A*. Also, the priority-stamp of i_A is 0, the highest (or smallest) priority value in the example.

For implementing the indirect causal relationships of simultaneous events in The time-and-priority-stamp, we propose the ε-delay scheme. The basic purpose of the ε-delay scheme is to insert a sufficiently small time delay artificially between the causing and resulting external events to simulate their indirect causal relationship; thus, the time-and-priority-stamp of the resulting external event becomes larger than that of the causing event. We formalize the ε-delay scheme by defining the *level* of a simultaneous event. The level of an event represents the relative position in the causal relationships to which the event belongs. Simultaneous events at a smaller level precede the ones at a larger level in their causal relationships. Also, the simultaneous events at the same level do not have any indirect causal relationships with each other. The rule for assigning a level to a simultaneous event is as follows. If an internal event and its causing external (or internal) event have different time-stamps, the level of the internal event is *0* (for example, the level of i_A is *0* in Figure 2). The level of an external event is the same as that of its causing internal event (for example, the level of x_{AB} is *0* (the level of i_A) in Figure 2). Finally, the level of an internal event is equal to the level of its causing event increased by one (for example, the level of i_B is $1+0$ (the level of x_{AB}) in Figure 2).

To implement the level of events in the time-and-priority-stamp, the ε-delay scheme is employed. The ε-delay is defined as follows.

definition 2 *The ε-delay is a sufficiently small positive time advance value, such that $\varepsilon * A < K$, for a sufficiently large constant A, where K is the minimum of any positive time advance values of all DEVS models to be simulated.*

When the level of an event is increased, the scheme increases the event's time-stamp value by an ε-delay. This scheme can be implemented easily in the abstract simulator algorithm[8]; that is, when the time advance function of a simulator returns *0*, the scheme simply replaces *0* with an ε-delay. Thus, the time-stamp of the resulting output event becomes always larger than that of the causing input event. Note that replacing zero time advance value with an ε-delay does not affect the simulation result since the ε-delay is a sufficiently small value compared to the time advance values specified by the modeler. For example, Figure 2 shows the resulting time-and-priority-stamps for our previous example. After processing the input external event x_{CE}, simulator *E* sets its next internal transition time to $9+\varepsilon$. Thus, the time-and-priority-stamps of i_E and x_{ED} become $\langle 9+\varepsilon, 0\rangle$ and $\langle 9+\varepsilon, 1\rangle$ respectively.

The ε-delay scheme is not the only method for

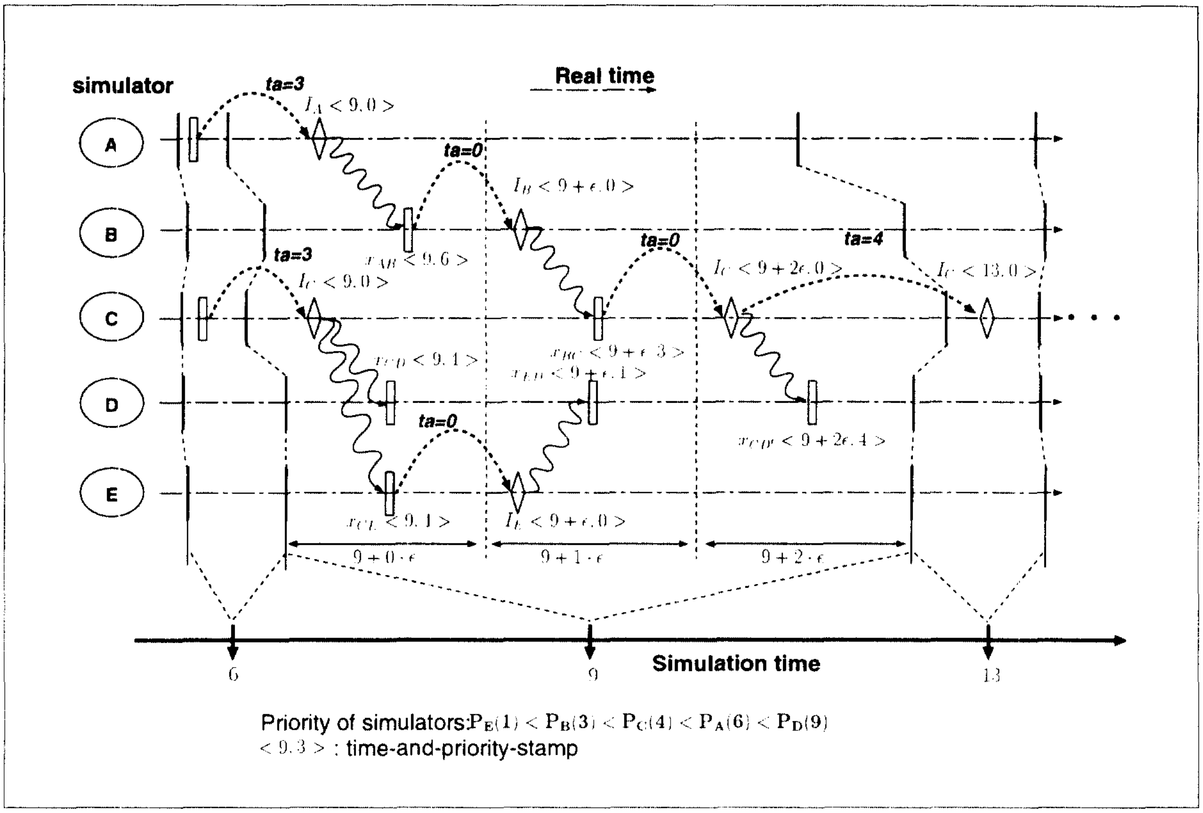
simulator
Real time
ta=3
$I_A < 9, 0 >$
ta=0
$I_B < 9 + \epsilon, 0 >$
$x_{AB} < 9, 6 >$
ta=3
$I_C < 9, 0 >$
ta=0
ta=4
$I_C < 9 + 2\epsilon, 0 >$
$I_C < 13, 0 >$
$x_{CD} < 9, 4 >$
$x_{BC} < 9 + \epsilon, 3 >$
$x_{ED} < 9 + \epsilon, 1 >$
ta=0
$x_{CD'} < 9 + 2\epsilon, 4 >$
$x_{CE} < 9, 4 >$
$I_E < 9 + \epsilon, 0 >$
$9 + 0 \cdot \epsilon$
$9 + 1 \cdot \epsilon$
$9 + 2 \cdot \epsilon$
A
B
C
D
E
6
9
13
Simulation time
Priority of simulators:$P_E(1) < P_B(3) < P_C(4) < P_A(6) < P_D(9)$
$< 9, 3 >$: time-and-priority-stamp

Figure 2 : Ordering simultoneous events vy the time-and-priority-stamp and∈-delay schems

implementing the level of events. For example, we can incorporate the level into the stamp of an event as a separate field; that is, the stamp of an event will consist of three fields: time, level, and priority. Then, these stamps can be ordered lexicographically. However, by employing the ϵ-delay scheme, the only use of the two fields, time and priority, becomes sufficient for such ordering. Note that the ϵ-delay is similar to the delta-delay in VHDL simulation [2]. Both delays represent the default propagation time of events in case there is not an explicit propagation time specified. Also, they are both small time delays greater than zero, but smaller than any explicitly specifiable time delay. However, they are employed for different purposes. The delta-delay is used for the simulation time advance mechanism of VHDL simulation ; thus, it should be used even in sequential VHDL simulation. In contrast, the ϵ-delay is employed to implement to the level in distributed simulation.

By employing the time-and-priority-stamp and ϵ-delay schemes, we can order totally all input events of a simulator while preserving their causal relationships. For example, the three external events of simulator *D*, *i.e.* x_{CD}, x_{ED}, and $x_{CD'}$, can be ordered correctly, as shown in Figure 2. Also, the schemes can make DEVS models meet the semantic rules of Time Warp as shown in

Theorem 1 *The proposed schemes can guarantee that DEVS models satisfy the semantic rules of Time Warp.*

proof : Rule 1 states that all input events of a simulator should be ordered totally. We can easily see that the first rule holds. Events with different time-stamps can be ordered easily by their time-stamps. Simultaneous events are partitioned into different levels. Simultaneous events with different levels have distinct time-stamps by the ε-delay scheme. Also, all simultaneous events with the same level have distinct priority-stamps since the source simulators of the events have distinct priorities by the SELECT function. Thus, all simultaneous events of a simulator can be ordered totally.

Rule 2 states that the send time-stamp of an external event should be smaller than its receive time-stamp. Note that an external event of DEVS models has only one time-and-priority-stamp since the sending and receiving of an event occur at the same simulation time in DEVS semantics, as mentioned before. However, we can consider that the send time-and-priority-stamp of an external event is equal to the time-and-priority-stamp of its causing internal event, since external events are generated during the execution of an internal event. From Definition 1, the time-and-priority-stamp of an internal event is smaller than that of external events generated from it (Note that the priority-stamp of an internal event is 0, the highest priority value). Thus, the second rule holds also, and this completes the proof. Therefore, the second condition holds also.

Conclusion

In this paper, we proposed a new events ordering mechanism in the optimistic distributed simulation of DEVS models. The proposed mechanism employs the time-and-priority-stamp and the ε-delay schemes. The mechanism can guarantee that DEVS models satisfy the semantic rules of Time Warp. Also, the mechanism can order all events correctly with respect to their causal relationships.

Reference

[1] J. R. Agre and P. A. Tinker. Useful extensions to a time warp simulation system. In *Advances in Parallel and Distributed Simulation*. pages 78-85. 1991

[2] James R. Armstrong and F. Gail Gray. *Structured Logic Design with VHDL*. Prentice Hall, 1993

[3] Alex C. Chow and Bernard P. Zeigler. Parallel DEVS: A parallel, hierarchical, modular modeling formalism. In *Proceedings of the 1994 Winter Simulation Conference*, Orlando, Florida, 1994.

[4] Eric R. Christensen and Bernard P. Zeigler. Distributed discrete event simulation : Combining DEVS and Time Warp. In *Proceedings of the SCS Eastern Multiconference on AI and Simulation Theory and Applications. Simulation Series, 1990.*

[5] Bruce A. Cota and Robert G. Sargent. An algorithm for parallel discrete event simulation using common memory. In 22th Annual Simulation Symposium. Simulation Series, 1989.

[6] David R. Jefferson. Virtual time. *ACM Transactions on Programming Languages and Systems*, 7(3) : 404-425, July 1985.

[7] Ki Hyung Kim. *Distributed Simulation Methodology Based on System Theoretic Formalism : An Asynchronous Approach.* PhD thesis, KAIST, 1996.

[8] Ki Hyung Kim, Yeong Rak Seong, Tag Gon Kim, and Kyu Ho Park. Distributed optimistic simulation of hierarchical DEVS models. In *Proceedings of the 1995 Fall Simulation Conference*, KAIST 1995. Korea Society for Simulation.

[9] Ki Hyung Kim, Yeong Rak Seong, Tag Gon Kim, and Kyu Ho Park. Distributed simulation of hierarchical DEVS models. In *Proceedings of the 1995 Fall Simulation Conference*, KAIST 1995.Korea Society for Simulation.

[10] Ki Hyung Kim, Young Rak Seong, Tag Gon Kim, and Kyu Ho Park. Distributed simulation of hierarchical DEVS models : Hierarchical scheduling locally and time warp globally. *Submitted to Transactions of the Society for Computer Simulation*, 1996.

[11] Ki Hyung Kim, Young Rak Seong, Tag Gon Kim, and Kyu Ho Park. Optimistic parallel simulation of hierarchical discrete event models. *Submitted to Parallel Processing Letters*, 1996.

[12] T. G. Kim and S . B. Park. The DEVS formalism :

Hierarchical modular systems specification in C++. In *Proceedings in European Simulation Multiconference*, 1992

[13] T. G. Kim and B. P. Zeigler. The DEVS scheme simulation and modelling environment. In *Chapter 2 in Knowledge Based Simulation : Methodolgy and Application*. Springer Verlag, Inc., 1990

[14] Leslie Lamport. Time, clocks, and the ordering of events in a distributedm system. *Communications of the ACM, 21(7):558-565, July 1978.*

[15] Horst Mehl. A deterministirc tie-breaking scheme for sequential and distributed simulation. In Proceedings of the 6th Workshop on Paralledl and Distributed Simulation, Pages 199-200, 1992.

[16] Jayadev Misra. Distributed discrete-event simulation. ACM *Computing Surveys*, 18(1):39-66, March 1986

[17] Rovert Sargent. Hierarchical modeling for discrete event simulation(panel). In *Proceedings of the 1993 Winter Simulation Conference*, page 569, Los Angeles, CA, 1993

[18] Yeong Rak Seong, Sung Hoon Jung, Tag Gon Kim, and Kyu Ho Park. Parallel simulation of hierarchical modular DEVS models : A modified Time Warp approach. *Iternational J. in Conputer Simulation*, 5(3): 263-285, 1995

[19] Yung Hsin Wang and Bernard P.Zeigler. Implementing the DEVS formalism on a parallel SIMD architecture. *International J. in Computer Simulation*, 5(3):229-245, 1995

[20] B. P. Zeigler. *Multifacetted Modeling and Discrete Event simulation. Academic Press, 1984*

[21] B. P. Zeigler and G. Zhang. Mapping hierarchical discrete event models to multiprocessor systems : Concepts, algorithm, and simulation. J. Parallel and Distributed Computing, 10(3):271-281, July 1990.

● 저자소개 ●

김기형
1990년 2월 : 한양대학교 공과대학 전자통신공학과 졸업(공학사)
1992년 2월 : 한국과학기술원 전기 및 전자공학과 졸업(공학석사)
1996년 8월 : 한국과학기술원 전기 및 전자공학과 졸업예정(공학박사)
관심분야 : 분산시뮬레이션, 시스템 분석 및 시뮬레이션, 멀티미디어 시스템, 실시간 고장허용 시스템

김탁곤
1975년 2월 부산대학교 전자공학과 졸업(공학사)
1980년 2월 경북대학교 전자공학과 졸업(공학서가)
1988년 5월 아리조나대학교 전기 및 전산 공학과 졸업(공학박사)
1975년 2월 1977년 6월 육군 통신장교
1980년 9월 1983년 1월 부산수산대학교 전자통신공학과 전임강사
1987년 8월 1989년 7월 아리조나 환경연구소 연구 엔지니어
1989년 8월 1991년 8월 캔사스대학교 전기 및 전산공학과 조교수
1991년 9월 1993년 8월 한국과학기술원 전기 및 전자공학과 조교수
1993년 9월 현재 한국과학기술원 전기 및 전자공학과 부교수 본학회 논문지

편집위원장
관심분야 : 모델링 이론, 병렬/지능형 시뮬레이션 환경, 컴퓨터시스템

박규호
1973년 서울대학교 전자공학과 졸업(공학사)
1975년 한국과학기술원 전기 및 전자공학과 졸업(공학석사)
1983년 프랑스 파리 11대학 졸업(공학박사)
1975년-1978년 동양정밀 개발과장
1983년-1988년 한국과학기술원 전기 및 전자공학과 조교수
1988년-1992년 한국과학기술원 전기 및 전자공학과 부교수
1992년-현재 한국과학기술원 전기 및 전자공학과 교수
관심분야 : 병렬처리, 컴퓨터구조, 컴퓨터그래픽스