Proceedings of the 2004 Winter Simulation Conference
R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

EXPLOITING TEMPORAL UNCERTAINTY IN PROCESS-ORIENTED DISTRIBUTED SIMULATIONS

Margaret L. Loper

Georgia Tech Research Institute
Georgia Institute of Technology
347 Ferst Drive
Atlanta, GA 30332, U.S.A.

Richard M. Fujimoto

College of Computing
Georgia Institute of Technology
Atlanta, GA 30332, U.S.A.

ABSTRACT

Existing research has defined a new type of simulation time called Approximate Time, where the simulation's knowledge about the values that represent time is uncertain. The approach is based on temporal uncertainty and uses time intervals rather than precise time values to represent time. Simulation language constructs are necessary to provide a convenient means of exploiting the temporal uncertainty to simulation modelers. To address this problem, a new time advance primitive for process-oriented simulations was developed, termed the Interval Hold construct. Interval Hold is an extension of the well-known hold primitive used in conventional simulation languages. This paper defines the interval time advance primitive and describes an algorithm for implementing it.

1 INTRODUCTION

The heart of every simulation is a time control program that advances simulation time and selects a subprogram to be executed (Kiviat 1969). In the Next Event time advance method, the clock is advanced to the time at which the next event is due to occur. Using this approach, each event is processed in a sequential manner. In parallel and distributed simulation environments this can lead to problems. If conservative execution is used, and the simulation has zero lookahead (which is often the case), a synchronization computation is required before each time advance to ensure no events arrive in an LP's past. The synchronization overhead and blocking imposed to ensure time stamp ordered event processing may prevent a simulation from achieving real-time performance in a distributed simulation environment. Exploiting temporal uncertainty provides a means of addressing this problem.

A natural, convenient means must be defined to specify uncertainty in simulation programs. To address this problem a time advance mechanism for process-oriented simulations called Interval Hold is proposed. Interval Hold uses Approximate Time clocks (Loper 2002) to increase the concurrency of event processing and improve the efficiency of the synchronization mechanism. The Interval Hold algorithm is important because it reduces the number of global synchronization computations required to causally order the events in a distributed simulation.

This paper starts by giving an overview of process-oriented simulation and Approximate Time. Following that, the interval time advance primitive is defined and an algorithm for implementing the primitive is described.

2 BACKGROUND

2.1 Process-Oriented Simulation

A process-oriented simulation is one of the three widely accepted modeling perspectives under which a simulation is developed (Lackner 1962, Kiviat 1967, Fishman 1973). A process-oriented simulation uses an abstraction called a process to model a specific object in the simulation with a well-defined behavior. A process is a time-ordered sequence of events, separated by passages of time, that define the life cycle of one entity as it moves through a system (Banks, Carson, and Nelson 1996; Law and Kelton 1991).

Process-oriented simulations are typically implemented on top of event-driven simulations, using the same event list and time advance mechanisms. The lifetime of a simulation process can be viewed as a miniature event-driven simulation in that it consists of a sequence of events; however, in process-oriented simulations, time only advances between these events. Typically a process-oriented simulation uses a "work" (CACI 1997), "wait" (CACI 1997), or "hold" (Meyer and Bagrodia 1999) primitive to advance time between the events. These primitives are essentially the same - they suspend a process from executing by placing an event into the event list with an event time that indicates the future time at which execution of the process should resume. When this event has the smallest time stamp in the event list, the simulation will advance its time and resume its execution.

The remainder of this paper uses the primitive Hold(T) as a general time advance mechanism for process-oriented simulations. The Hold primitive causes simulation time

for the process to advance by T time units. The primitive is invoked to signify that the simulation object is busy performing some activity for T units of time.

2.2 Approximate Time

Approximate Time is based on temporal uncertainty and uses time intervals rather than precise time values to represent time. An interval is a closed bounded set of "real" numbers (Moore 1979) [ET, LT] = {t: ET ≤ t ≤ LT}, where ET denotes the earliest time (or E-time) in the interval and LT the latest time (or L-time). An overview of Approximate Clocks, the mechanism for time stamping events and maintaining the simulation's notion of approximate time and Approximate Time Causal (ATC) order, the temporal precedence order used in approximate time can be found in (Loper 2002, Fujimoto 1999).

3 INTERVAL HOLD PRIMITIVE

This section describes a new time advance primitive that provides a way for simulations to specify temporal uncertainty in order to exploit the ATC ordering algorithm described in (Loper 2002, Fujimoto 1999). The approach is called Interval Hold (iHold). It allows the simulation to specify an interval instead of a precise time stamp as the parameter for time advances. The interval indicates the uncertainty in how far into the future the simulation wishes to advance. The set of concurrent events that overlap the time advance interval can safely be delivered, thus increasing the concurrency and improving performance of the synchronization algorithm.

The Interval Hold primitive is an extension of the Hold primitive, but uses a time interval rather than precise timestamp for its time parameter. The primitive is specified as iHold(ET, LT), where [ET, LT] indicates uncertainty in how long the entity will be busy performing its activity. The leading edge of the interval ET is the earliest time the LP should resume from the iHold, and the trailing edge LT is the latest time it should resume. In other words, the simulation would like to advance at least ET time units but not more than LT. The size of the interval is based on the specifics of the simulation; the larger the interval, the more uncertainty associated with the amount of time required to perform the activity. There are two constraints on the time advance interval. First, the leading edge of the interval ET must be less than or equal to the trailing edge LT, i.e. ET ≤ LT. Second, the leading edge of the interval ET must be greater than or equal to the simulation's current time plus lookahead, if the process lookahead is not zero (the remainder of this paper assumes that the lookahead is zero). These constraints ensure that the simulation does not send or receive events in the past.

There are two types of iHold that may be used in a simulation: 1) Hold for a certain amount of time or 2) Hold for a certain amount of time, unless something else happens. These two approaches to iHold can be thought of as *non-interruptible* and *interruptible*, respectively. In the non-interruptible iHold, the simulation will hold between ET and LT time units. Any new messages received during the iHold will be delivered after the iHold completes. The interruptible iHold will hold for the requested amount of time (i.e., [ET, LT]) unless a new message is received. In this case, the iHold will be interrupted so that the message can be delivered.

When iHold completes or is interrupted for event delivery, the simulation's time is advanced. This is known as the resume time. The resume time indicates that no events will later be delivered to the process with a time stamp less than the time of the resume. Once the process invokes iHold, it guarantees that it won't generate a new event with a time stamp less than or equal to LT (or less than or equal to LT plus lookahead if the process lookahead is not zero) until it has received a resume time (this paper assumes that precise time clocks are used, meaning that both events and the resume time are assigned precise time values within the interval).

There are many existing event delivery mechanisms that can be used for iHold, e.g. a specific call (Meyer and Bagrodia 1999) where the process asks to receive events or a call back (IEEE 2001) which notifies the process of awaiting events.

3.1 Non-Interruptible iHold

In the non-interruptible iHold, the process blocks for the requested amount of time (between [ET, LT]). Any new events received by the process will be stored in a message buffer. When the process resumes from the iHold, any events in the buffer will be delivered. The process will advance its simulation time to a value T, where ET ≤ T ≤ LT. When the process receives a resume time, it guarantees that it won't generate a new event with a time stamp less than the resume time (or less than the resume time plus lookahead if the process lookahead is not zero).

In the example below, the statement iHold(ET, LT, msg_type) is used to initiate an interval hold. The msg_type parameter indicates the type of message for which iHold can be interrupted. If the parameter is NULL, the iHold is non-interruptible; if the parameter is not NULL, then iHold can be interrupted for the type of message specified in the parameter. As described above, the receive statement is used for event delivery.

```
/* non-interruptible iHold */
iHold (ET, LT, NULL)
receive (gateInfo)
```

In this example, the iHold primitive specified a NULL message parameter indicating that it could not be interrupted. Therefore, the process receives a resume time T, where ET ≤ T ≤ LT. Upon the resume, the receive statement is then used to receive a `gateInfo` message.

3.2 Interruptible iHold

If the process is interruptible, an event with a timestamp less than or equal to LT is eligible for delivery during the iHold. The following events will be delivered to the process:

- Any event with a time stamp less than ET will be delivered
- If the event has a time stamp TS such that ET ≤ TS ≤ LT, this event may be delivered before simulation time is advanced, but it is not guaranteed to be delivered on this call to iHold
- No event with TS > LT will be delivered.

If a process receives an event before the iHold completes, the process will advance its simulation time to the time stamp of the delivered event. When the process receives a resume time, it guarantees that it won't generate a new event with a time stamp less than the resume time (or less than the resume time plus lookahead if the process lookahead is not zero).

In the example below, the interruptible iHold is shown. The iHold statement specifies a message type, indicating that it should be interrupted if a `departure-Info` message arrives. The receive statement follows the iHold so the message can be received by the process.

```
/* interruptible iHold */
iHold(ET, LT, departureInfo)
receive (departureInfo)
```

3.3 Example

Consider an example to illustrate how Interval Hold works. The simulation is of air traffic arriving and departing at an airport, as described in (Fujimoto 2000). The simulation models many planes; each one is a separate process. Upon arrival, each aircraft must: (1) wait for the runway and land, (2) travel to the gate and unload and load new passengers, and (3) depart and travel to another airport. In a precise world, the times required for (1), (2) and (3) would be fixed. However, there could be uncertainty associated with each event. For example, the time required to wait for the runway and land depends on such factors as weather and time of day. Similarly, the time spent at the gate unloading and loading depends on factors such as the number of passengers and the duration of the flight. These events will definitely happen, but there is uncertainty as to exactly when. In the case where there is some minimum and maximum time associated with each event, but uncertainty as to precise instant, iHold is a useful modeling approach. The pseudo code for this simulation example is shown in Figure 1.

The aircraft will first send a message to the controller requesting the runway. The aircraft then waits to receive a `Runway_Free` message from the controller indicating it is

```
/* Message Types */
message Request_Runway { int id; }
message Runway_Free { int id; }
message Gate_Info { int number; }
message Depart_Info { real delay; }
message Departure { int id; }

/* Process Aircraft */
entity Aircraft ()
{
    /* aircraft arrival, circling, and landing */
    id = my_plane_id;
    send Request_Runway{id} to controller /*arrival*/

    iHold (0, 0, Runway_Free)      /* circle */
    receive (Runway_Free runwayfree)
    iHold(ET_R, LT_R, NULL);       /* land */
    receive(Gate_Info gatenumber);
    send runwayfree to controller;

    /* simulate aircraft on the ground */
    iHold (ET_G, LT_G, Depart_Info)
    receive (Depart_Info delaytime)
    send Departure {id} to controller;  /* departure */
}
```

Figure 1: Simulation Program for a Single Airport

safe to land. This type of hold is based only on message type; there is no time parameter. In other words, the aircraft must continue to hold (however long it takes) until it receives approval to land. This type of hold is accomplished by invoking iHold(0, 0, `Runway_Free`). By setting the iHold time parameter [ET, LT] to zero, the iHold will be based on message type only. The receive message is then used to deliver the `Runway_Free` message to the process.

Once the aircraft has approval to land, it will invoke iHold (ET_R, LT_R, NULL) to indicate it is busy landing. In other words, the plane will use the runway for at least ET_R and not more than LT_R time units. This is a non-interruptible iHold. When the iHold resumes, the process advances its simulation time and invokes a receive primitive to deliver any `Gate_Info` messages that arrived while it was landing. The aircraft completes the landing process by sending the controller a `Runway_Free` message indicating it has cleared the runway.

The next step is to simulate the aircraft on the ground. This is accomplished with an interruptible iHold, by invoking iHold (ET_G, LT_G, `Depart_Info`). The time specified (ET_G, LT_G) indicates the time required at the gate to unload and load passengers. The `Depart_Info` message contains any last minute information for the aircraft before it departs the airport. If a `Depart_Info` message arrives while the aircraft is at the gate, then the message will be delivered to the aircraft. If no `Depart_Info` message is

received, the iHold will complete, and the plane sends a Departure message to the controller indicating it is departing the airport.

4 THE INTERVAL HOLD ALGORITHM

In the previous section the iHold primitive was described and the issues associated with using an interval as the time parameter discussed. In this section an algorithm for implementing iHold and selecting the resume time will be presented.

4.1 Sequential iHold

The iHold primitive is implemented using a local event a simulation schedules for itself containing an interval time stamp. This event is placed in the simulations local event list and ordered according to the LT of the events. When the simulation starts its main loop, it must find the smallest time stamped event among those scheduled in its local event list. An algorithm for implementing Interval Hold is shown in Figure 2.

P_i = set of scheduled events in Process i

for (each $X \in P_i$ with a precise timestamp)
  $E(X) = T(X)$
  $L(X) = T(X)$
$ELT = \min (L(X))$ for all $X \in P_i$
$S_i = \{\text{all events } Y: E(Y) \le ELT \le L(Y)\}$
$EET = \min (E(X))$ for all $X \in S_i$
$LET = \max (E(X))$ for all $X \in S_i$
resume time = T, where $EET \le T \le ELT$

Figure 2: Interval Hold Algorithm

There are two possible combinations of scheduled events: events with interval time stamps and events with both precise and interval time stamps. When only events with interval time stamps are scheduled, the minimum time interval among the scheduled events must be computed which determines the time to advance the logical clock. The other possibility is that there is a mixture of precise and interval time stamps in the local events list. In this case, the precise timestamps are converted to intervals (Kaufmann and Gupta 1991) and the minimum time interval is computed among the scheduled events.

By successively comparing the event intervals, the earliest ET (EET) and earliest LT (ELT) can be computed as the minimum time interval, as shown in Figure 3. The ELT value defines the latest time to which the simulation can advance its logical clock. If the simulation sets its clock to a value greater than ELT, it may violate ordering constraints. The EET value defines the earliest time the simulation should set its local clock. If the simulation sets its clock to a time less than EET, no progress can be made since there are no events to process.

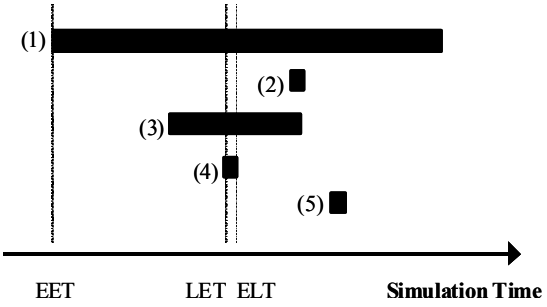

Figure 3: Computing the Minimum Time Interval

It is important to note that if more than one event is processed concurrently, they must both receive the same time stamp. Otherwise the earliest event processed could generate a new event with a timestamp less than the later event. This would clearly violate the ordering constraint. Also, the precise time stamp assigned to the event when it is processed can be any value greater than the leading edge of the event. For example, the value LET could be assigned to events (1), (3) and (4) or a value > LET. The important thing is that all the events processed receive the same time stamp.

4.2 Distributed iHold

Synchronization assures the temporal ordering of events. In a parallel or distributed simulation, the key to synchronization is a quantity called the lower bound on time stamp (LBTS) for each process. LBTS is the smallest timestamp of any event a process will receive in the future; if simulations have zero lookahead, it is equivalent to the minimum time stamp of any unprocessed or partially processed event in the system. The time stamp of any unprocessed event is a lower bound on future events that may be produced after processing that event. In other words, the event with the smallest time stamp (T = TS) could affect every simulation at time T. To compute LBTS, a snapshot of the computation (including messages in transit) is required so that the global minimum time can be computed. Once LBTS has been computed, all TSO messages containing a time stamp less than LBTS can be delivered.

If simulations in a distributed computation use the iHold primitive to advance time, their contribution to the LBTS computation will be a minimum time interval [EET_{local}, ELT_{local}]. Therefore, the key to using iHold in a distributed simulation is the ability to compute an interval LBTS value. Interval LBTS is the minimum time interval [LET_{global}, ELT_{global}] from the set of pending local minimum time values computed by each simulation. The LBTS interval identifies the set of processes that are safe to advance their clocks concurrently, and the range of time values those processes can be granted to without causing

messages to be delivered in the past. The interval LBTS algorithm is described in (Loper 2002 and Fujimoto 1999).

There are several existing delivery mechanisms that can be used with the iHold algorithm. To illustrate how events could be delivered in a simulation using distributed iHold, consider the algorithm shown in Figure 4.

T_{msg} = event in the simulations TSO queue with the minimum time stamp

if (($T_{msg} = LET_{global}$) && ($T_{msg} \leq ELT_{local}$))
    deliver message
    resume time = T_{msg}
else if (($EET_{local} \leq ELT_{global}$) && ($ELT_{local} \geq ELT_{global}$))
    resume time = T, where $LET_{global} \leq T \leq ELT_{local}$
else
    EET_{local}= min ET in local event list
    ELT_{local}= min LT in local event list
    Start iLBTS with minimum time =(EET_{local}, ELT_{local})

Figure 4: Delivering Events and Resume Times

There are three evaluations that take place in this algorithm. First, in order to release a message to the process, the time stamp of the message (T_{msg}) must be safe to execute and the process must be ready to receive messages for this time. This means that T_{msg} must be less than the LBTS value and less than the minimum time interval [EET_{local}, ELT_{local}] provided by the simulation. From the definition of the LBTS interval [LET_{global}, ELT_{global}], LET_{global} is the minimum LBTS time value. And according to the iHold definition, a message with a timestamp less than the specified ET is guaranteed delivery. Therefore, since LET_{global} is the largest of all ET values, a message with a timestamp less than or equal to LET_{global} will be delivered to a simulation. The simulation will then advance its time to the time stamp of the message delivered.

If there are no messages in the process's queue that are safe to execute, the second evaluation determines if a process's simulation time can be advanced to a time specified in the iHold interval. In order to advance the simulation's logical clock, the minimum time interval provided by the simulation must be less than LBTS. To make this evaluation, we compare two intervals: iHold and LBTS. According to (Allen 1983) there are seven possible relationships between two intervals, thirteen counting the inverses of the relations, as shown in Figure 5.

Of the thirteen possible relations, the cases where [EET_{local}, ELT_{local}] $\leq$ LBTS must be identifed. If $EET_{local} > ELT_{global}$ then there is no part of [EET_{local}, ELT_{local}] that precedes the LBTS interval. Therefore, these cases can be eliminated. This includes relation 2 from Figure 5. Also the ELT_{local} cannot be less than ELT_{global} as proven in (Loper 2002). This includes cases 1, 4, 6, 8 and 11. Further 7, the EET_{local} cannot be greater than LET_{global}, which includes cases 7 and 13. The remaining five relations for [EET_{local}, ELT_{local}] and [LET_{global}, ELT_{global}] are shown in Figure 6.

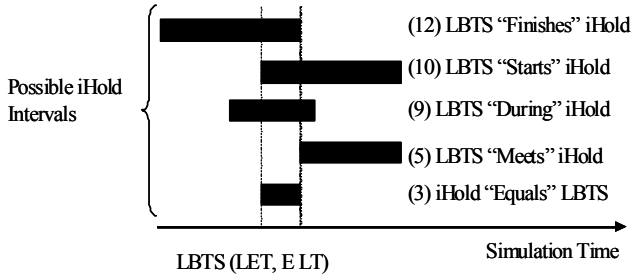

Figure 6: Interval Relationship Where ET Precedes ELT

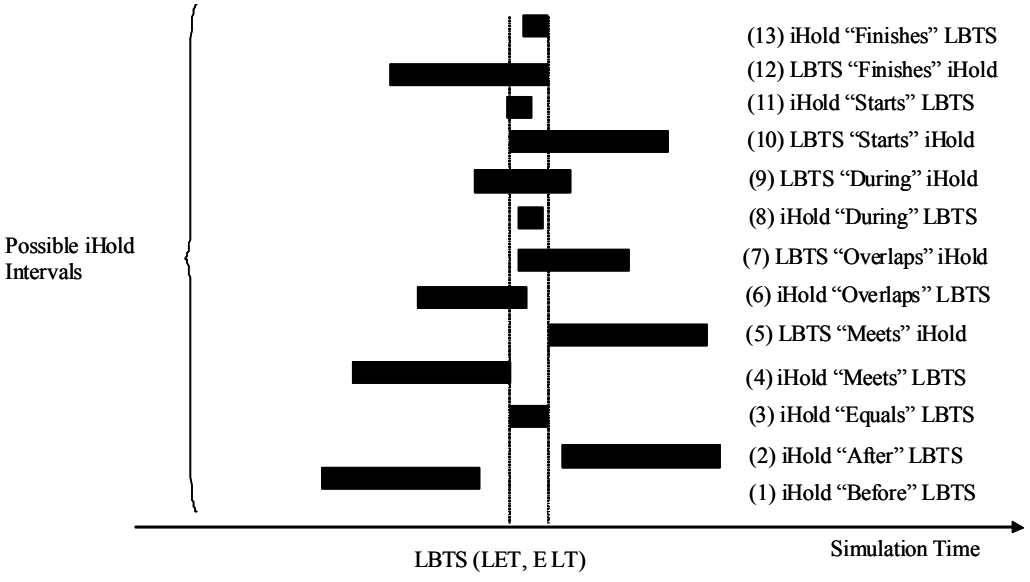

Figure 5: Relationship of LBTS Interval and iHold Interval

As proven in (Loper 2002), a simulation that has an interval $[EET_{local}, ELT_{local}]$ that intersects ELT_{local} is considered a concurrent simulation and can be returned a resume time in the LBTS interval, i.e. $LET_{global} \leq T \leq ELT_{global}$.

Finally, if no messages can be delivered and the process's simulation time cannot be advanced a new LBTS computation must be started. In order to start or respond to an LBTS computation, each process must compute its local minimum time interval $[EET_{local}, ELT_{local}]$.

5 CONCLUSIONS

The heart of every simulation is a time control program (Kiviat 1969). In this paper, a new time advance primitive for process-oriented simulations was developed, termed the Interval Hold construct. Interval Hold uses Approximate Time clocks to increase the concurrency of event processing. In (Nance 1971) a continuum of algorithms for representing the passage of time was presented. It proposed that fixed time increment defines one end of a continuum and next event increment the other. Between the two extremes are algorithms that possess characteristics of both which may be better suited for specific discrete system simulations. Nance went on to say that, in many cases, the efficiency of a simulation program's execution rests primarily with the procedure for incrementing time. By varying the size of the iHold interval, the interval time advance becomes a mechanism that spans the continuum of time advance algorithms discussed in (Nance 1971).

ACKNOWLEDGMENTS

Funding for this research was provided under the DARPA Advanced Simulation Technology Thrust (ASTT) program, the Link Foundation Fellowship in Advanced Simulation and Training, the NASA Office of Space Science GSRP Fellowship, and the GTRI graduate assistance program.

REFERENCES

Allen, J. 1983. Maintaining Knowledge about Temporal Intervals. *Communications of the ACM*, 26 (11): 832-843.

Banks, J., J.S. Carson II, and B.L. Nelson. 1996. *Discrete-Event System Simulation*. N.J.: Prentice Hall.

CACI. 1997. Simscript II.5 Programming Language, 210-211. CACI Products Company, La Jolla, CA. Available online via <http://www.caciasl.com/cust_center/ ss3docs/simprog.pdf> [accessed August 20, 2004].

Fishman, G. 1973. *Concepts and Methods in Discrete Event Digital Simulation*. New York: John Wiley.

Fujimoto, R.M. 1999. Exploiting Temporal Uncertainty in Parallel and Distributed Simulations. In *Proceedings of the 13th Workshop on Parallel and Distributed Simulation*, ed. R. Fujimoto and S. Turner, 46-53. Piscataway. New Jersey: Institute of Electrical and Electronics Engineers.

Fujimoto, R. 2000 *Parallel and Distributed Simulation Systems*. Parallel and Distributed Computing, ed. A.Y. Zomaya. New York: John Wiley & Sons, Inc.

IEEE. 2001. IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) - Federate Interface Specification. IEEE: New York, NY.

Kaufmann, A. and M.M. Gupta. 1991. *Introduction to Fuzzy Arithmetic Theory and Applications*. London: International Thomson Computer Press.

Kiviat, P. 1967. Digital Computer Simulation: Modeling Concepts. RAND: Santa Monica, CA.

Kiviat, P. 1969. Digital Computer Simulation: Computer Programming Languages: Santa Monica, CA.

Lackner, M.R. 1962. Toward a General Simulation Capability. In *Proceedings of the AFIPS Spring Joint Computer Conference*, 1-14. San Francisco, CA.

Law, A.M. and W.D. Kelton. 1991. *Simulation Modeling & Analysis*. McGraw-Hill, Inc.

Loper, M.L.. 2002. Approximate Time and Temporal Uncertainty in Parallel and Distributed Simulation. Doctoral dissertation, College of Computing, Georgia Institute of Technology, Atlanta, Georgia.

Meyer, R.A. and R. Bagrodia, 1999. PARSEC User Manual, 11. Parallel Computing Laboratory, Department of Computer Science, UCLA: Los Angeles, CA. Available online via <http://pcl.cs.ucla.edu/projects/parsec/manual.pdf> [accessed August 20, 2004].

Moore, R.E.. 1979. *Methods and Applications of Interval Analysis*. SIAM Studies in Applied Mathematics, ed. W. Ames. Philadelphia, PA: Society for Industrial and Applied Mathematics.

Nance, R. 1971. On Time Flow Mechanisms for Discrete System Simulation. *Management Science* 18(1): 59-73.

AUTHOR BIOGRAPHIES

MARGARET LOPER is a Senior Research Scientist at the Georgia Tech Research Institute (GTRI) at the Georgia Institute of Technology. She earned a B.S. in Electrical Engineering from Clemson University in 1985, an M.S. in Computer Engineering from the University of Central Florida in 1991, and a Ph.D. in Computer Science from the Georgia Institute of Technology in 2002. Her current research interests include synchronization algorithms, temporal uncertainty, parallel and distributed systems, and theoretical aspects of modeling. Her research has been funded by DARPA, the Defense Modeling and Simulation Office, and the U.S. Army STRICOM. Her e-mail address is <margaret@cc.gatech.edu> and her website is <http://www.cc.gatech.edu/~margaret/>.

RICHARD FUJIMOTO has been working in the area of parallel and distributed simulation systems and environments since 1985. He is the principal architect of the Georgia Tech Time Warp (GTW) parallel/distributed simulation

executive that has been used to model telecommunication networks, aviation and military systems. He has also worked on developing distributed simulation software to facilitate interoperability and reuse of simulations, and lead the development of the RTI-Kit software for realizing high-speed distributed simulation systems. Fujimoto chaired the working group responsible for defining the time management services for the High Level Architecture (HLA) that has been designated as the standard architecture for modeling and simulation in the U.S. Department of Defense. He is an area editor for ACM Transactions on Modeling and Computer Simulation and Co-Editor-in-Chief of SCS Transactions His e-mail address is <fujimoto@cc.gatech.edu> and his website is <http://www.cc.gatech.edu/~fujimoto/>.