ZERO LOOKAHEAD AND REPEATABILITY IN THE HIGH LEVEL ARCHITECTURE

Richard M. Fujimoto
College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

KEYWORDS
High Level Architecture, Run-Time Infrastructure, time management

ABSTRACT

A distributed simulation is said to be repeatable if successive executions utilizing the same inputs produce exactly the same outputs. Repeatability is a highly desirable property, particularly for analytic simulation models. This paper discusses the question of repeatability in distributed simulations in general, and in the context of the High Level Architecture in particular. Specifically, allowing zero lookahead, a feature not supported in the baseline HLA, has important ramifications with respect to achieving repeatable executions. Extensions to the existing time management services in the HLA are proposed that allow (1) repeatable executions, (2) federate control over the ordering of simultaneous events, and (3) zero lookahead. The extensions proposed in this paper are currently under consideration by DMSO for possible future inclusion in the High Level Architecture time management services.

1. INTRODUCTION

It is sometimes important that repeated executions of a simulation program using the same inputs and initial conditions (random number seeds, etc.) produce exactly the same results as previous executions, particularly in simulations used for analysis. This may be necessary because independent agencies need to re-run simulation programs to verify previously reported results. Repeatability is also helpful in debugging the simulation program because it facilitates identification and correction of bugs and erroneous results.

It is usually straight-forward, if not trivial, to achieve repeatability in *sequential* simulation programs, at least if the same computing hardware is used for each execution. In general, one need only ensure individual computations do not perform non-repeatable operations such as reading the real-time clock, and ensure the same external inputs are provided to the simulation program. If the computing platform changes from one execution to the next, one must also ensure that machine operations (e.g., floating point round-off) yield the same results across the different platforms.

Repeatability is not as easily obtained in a distributed simulation, however, even if the individual simulation computations within each processor are repeatable. In training simulations (e.g., using DIS protocols), many factors prevent the repeatability of simulation executions. A few obvious ones include the use of best effort message delivery (different messages will be lost from one execution to the next), receive ordered processing of events (messages will usually be received in a different order from one execution to the next), and lack of repeatability for interactive inputs. Repeatability can, in effect, be achieved by producing a log of events during an initial, *reference* execution, and utilizing this log in subsequent executions to ensure events are processed in the same order using the same data from one execution to the next.

Achieving repeatability is more straightforward in distributed simulations utilizing logical time. Logical time simulations typically use reliable message delivery, and external inputs can be applied in a repeatable fashion by ensuring the same time stamp is assigned to each external input in subsequent executions. The computation performed by each simulator (federate, in HLA terminology) can be viewed as a sequence of sub-computations, one for each event generated locally or by other federates. A key observation that greatly facilitates repeatability is events are processed in time stamp order. Thus, ignoring events containing the same time stamp, one need only ensure each event computation is repeatable, thereby ensuring the same events will be created in each execution, to ensure repeatability of the distributed simulation.

In addition to processing events in time stamp order, one must also ensure that events containing the same time stamp are processed in the same order on each execution to ensure repeatability. Moreover, the treatment of these so-called *simultaneous events* must be handled in a way that satisfies the requirements imposed by modelers developing the simulation program. There may be causal relationships between simultaneous events, e.g., processing an event at time T may result in scheduling a new event with the same time stamp. This paper examines these issues, and describes the approach now being explored by DMSO for inclusion in the DoD High Level Architecture.

This paper focuses exclusively on repeatability in logical time-based distributed simulations. The next sections define terminology used throughout this paper and review related work. Requirements on the HLA approach to repeatability, as expressed by protofederations involved in experimentation with the HLA are described. The sections that follow describe the approach currently defined in the HLA to achieving repeatability assuming non-zero lookahead, and proposed extensions to this approach where zero lookahead is allowed.

2. DEFINITIONS

A distributed simulation is said to be *repeatable* if subsequent executions of the simulation using the same initial conditions and input as a "reference" execution produce exactly the same results (e.g., model statistics) as the reference execution. *Simultaneous events* refer to two or more events containing the same time stamp.

Distributed logical time simulations in general, and specifically those defined in the HLA, utilize a non-negative quantity called *lookahead.* If a federate at logical time T has a lookahead of L, any event scheduled by the federate must have a time stamp of at least T+L. Lookahead is used to define a lower bound on the time stamp of messages produced by a simulator later during the execution (T+L in this example) since the current logical time of a simulator can never decrease. By computing a minimum of this lower bound across all simulators that can send messages to another simulator S, one can derive a lower bound on the time stamp of messages that will be delivered to S in the future. This lower bound is important because it means S can process messages with time stamp smaller than this lower bound without fear of later receiving a smaller time stamped message, which would violate the constraint that all events be processed in time stamp order. The HLA currently calls for each federate to define a positive (i.e., non-zero) value of lookahead[1].

It is assumed that time stamp ordered and reliable delivery (as defined in [2]) are used for all messages during the federation execution. Further, it is assumed that individual computations performed by each simulator in the distributed simulation are repeatable, at least to the extent that these computations affect the results produced by the distributed simulation.

3. RELATED WORK

Repeatability may be accomplished by using a tie-breaking mechanism to ensure repeatable ordering of simultaneous events. Two related issues are user control over the ordering of simultaneous events, and whether or not zero lookahead is allowed. Related work concerning each of these subjects are discussed next.

3.1 Tie-Breaking Mechanisms

Work in the parallel discrete event simulation community concerning repeatability has been primarily concerned with developing repeatable tie breaking mechanisms to ensure simultaneous events are ordered in a repeatable (or sometimes called deterministic) fashion[3]. This is typically accomplished by the underlying distributed simulation executive appending additional, lower precision bits to the time stamp of each event in order to ensure, in effect, unique time stamps. Events will always be processed in time stamp order, where the time stamp now includes these tie-breaking bits, to ensure that simultaneous events are always processed in the same order. An important requirement of the tie-breaking bits is if event A schedules event B, then the time stamp for A must be less than that of B, where the time stamp includes the tie breaking bits.

Mehl proposes that the RTI append two tie-breaking fields to the application defined time stamp called the *age* and *id*, with the age field given precedence (assigned to more significant bits) than the *id* field[3]. Events that exist at the beginning of the simulation are assigned an age of 1. If an event with time stamp T and age *A* schedules another event with the same time stamp (ignoring the extra fields), the new event is assigned an age of *A+1*. If the new event has a time stamp larger than T, the age of the new event is 1.

The *age* field ensures events always schedule other events with higher time stamps, but does not ensure uniqueness, e.g., an event with time stamp T, and age 5, could schedule two new events also with time stamp T. In this case, both events will have an age of 6. The *id* field ensures uniqueness. This field is actually a tuple with two components (S, i).[1] S is the ID of the source process (simulator) generating the event. It is assumed process IDs remain the same from one execution to the next. The *i* field is a counter indicating this is the ith event scheduled by the process.[2]

Schemes similar to this have been implemented in other parallel simulation executives. For example, the Time Warp Operating System (TWOS) developed at JPL [4] and SAIC's Tempo system[5] use similar approaches. The TWOS implementation uses the body of the event itself (including parameters and other fields) for the *id* field[4]. Though not guaranteeing unique time stamps, this approach guarantees that only identical events will have the same time stamp, in which case ordering is irrelevant.

[1] Mehl actually proposes a triple, (S, i, R) where R is the receiving process, however, this field is redundant, so is ignored here.

[2] In optimistic synchronization mechanisms such as Time Warp, the *i* field only includes events that are not later rolled back. This can be realized by including the counter among the variables that are state saved, and rolled back.

3.2 User Controlled Ordering of Simultaneous Events

It is important that the model developer be able to control the ordering of simultaneous events. This is not taken into account by the tie breaking mechanisms described in the previous section, which are implemented in the simulation executive (or RTI), beyond the reach of the modeler. To illustrate this point, consider two simultaneous missile detonation events at a target. The order that these events are processed affects which missile is credited with the kill. An RTI that arbitrarily determines the order of these events might systematically favor one weapon system over another.

The importance of ordering simultaneous events has long been recognized in the discrete event simulation community. For example, this issue is discussed extensively in [6] and several examples are described explicitly specifying the ordering of simultaneous events. Examples where processing simultaneous events in an incorrect order leads to semantic errors are discussed in [7, 8]. Wieland observes that statistics in a queueing network simulation vary significantly using different rules for breaking ties, and discusses extension of the simultaneous event ordering problem to events containing time stamps that are close, but not identical[9].

3.3 Priority Fields

Priorities are sometimes used to specify the ordering of simultaneous events. For example, a priority number might be assigned to the event with the semantics that an event with a smaller priority number will be processed before other simultaneous events containing larger priority numbers. If a simulator schedules a zero lookahead event (an event with time stamp equal to the logical time of the scheduler), the priority number of the new event must be at least as large as that of the current event. Operationally, this might be implemented by appending the priority number to the time stamp so that it has lower precedence than the application defined time stamp, but higher precedence than the bits used to resolve ties[3].

The main drawback with this approach is that it requires the simulator *scheduling* the event to assign the priority, but in most cases, it is more convenient to have the simulator receiving the events to order them. This is because the receiver can more easily determine what other events occur at the same logical time, and the order in which the events should be processed may depend on the state of the receiver. If the ordering depends on the state of the scheduler, the necessary information can be easily included within the event itself.

3.4 Zero Lookahead

A simulator at logical time T can only schedule a new event with time stamp T if zero lookahead is allowed. Thus, whether or not the simulation system allows the scheduling of zero lookahead events impacts the approach that is used to handle simultaneous events. This will be discussed in greater detail later in the context of the HLA.

From the perspective of the simulation modeler, zero lookahead events are useful for several reasons:

- They can be used to implement query events. Zero lookahead allows one simulator to query another simulator for information, and receive a response *without advancing its local simulation time*. For example, once a sensor has detected a vehicle has moved within its range, the simulator for the sensor might query the vehicle simulator to collect information such as the type of vehicle, whether or not it is a threat, etc.
- Zero lookahead can be used if the actual advance in simulation time is so small that it is less than the precision of time stamp values. For example, the precision of time stamp values might be in minutes, while the required logical time advance is a few seconds.
- Initial, simplified models might use zero lookahead prior to more detailed specification of temporal characteristics.
- A single event in the actual system might be modeled as several, distinct, simultaneous events in the simulation. For example, a ship contacting a mine (from the perspective of the ship) and the mine being struck by the ship (from the perspective of the mine) might be a single action in the actual system, but could be modeled by two distinct simultaneous events that are realized by one event causing the second to be scheduled.

Certain simulation protocols place constraints on whether or not zero lookahead is allowed. The Chandy/Misra/Bryant null message protocol[10] does not allow a cycle of processes with zero lookahead to exist, as this will result in an unending cycle of null messages containing the same time stamp. Other simulation protocols, e.g., the deadlock detection and recovery protocol developed by Chandy and Misra[11] do allow zero lookahead. Jha and Bagrodia discuss lookahead constraints for several different simulation protocols in order to implement arbitrary user defined ordering of simultaneous events[12].

4. REPEATABILITY IN THE HIGH LEVEL ARCHITECTURE

The current definition of the HLA time management services described in [2] support repeatable federation executions, but requires non-zero lookahead. Protofederations using the initial implementation of the HLA have since expressed the desire to allow zero lookahead. Below, we review the requirements that have been expressed for the time management services with respect to repeatability, lookahead, and simultaneous events. We then describe the current approach used in the HLA to achieve repeatability (but relying on zero lookahead), and describe additions to the HLA to allow repeatability in the presence of federates with zero lookahead.

4.1 Requirements

The HLA should conform to the following requirements:

Requirement 1: Repeatability. It must be possible to build federations in the HLA where execution is repeatable. This is important because outside agencies such as the General Accounting Office must be able to re-run simulations to verify results that are used for procurement purposes. It should be noted that the RTI cannot *guarantee* repeatability of federation executions, it can only facilitate construction of federations where execution is repeatable.

Requirement 2: Federation control in ordering simultaneous events. The proper ordering of simultaneous events must be decided within the federates, not the RTI. This is because the appropriate ordering usually depends on interpretation of the meaning of the events. This issue is especially pronounced in time-stepped simulations where many events may occur that contain the same time stamp. The RTI cannot make meaningful ordering decisions because it does not have knowledge concerning the semantics of the events.

Requirement 3: Zero Lookahead. Federates must be allowed to have a lookahead value of zero. The specific use of zero lookahead that has been cited is to allow *query events* where a federate at logical time T can request the value of an attribute from another federate at time T, and receive the response while remaining at logical time T. This approach is commonly referred to as "pull processing".

Requirement 4: Avoidance of "logical time creep." "Logical time creep" is a phenomenon where logical time throughout the federation advances very slowly when there are no events in a certain interval of logical time. It is well known that the Chandy/Misra/Bryant algorithm[10] is susceptible to this phenomenon (e.g., see [13]). To illustrate this problem, consider the situation where all federates are at logical time 10, each federate has a lookahead of 1, and the smallest time stamped event in the entire federation has a time stamp value of 100. The CMB algorithm will advance the logical time of each federate to 11, 12, 13, ... 99, 100 (more generally, in steps of the lookahead size), with each increment to the next logical time value only occurring after NULL messages are exchanged among the RTIs. This problem is easily eliminated by the RTI recognizing that the next event has time stamp 100, and immediately advancing all of the federates to logical time 100. Several synchronization algorithms are described in the parallel discrete event simulation literature exploiting this fact (e.g., [11] was perhaps the first, see[14] for others). It should be noted, however, that even with the use of more advanced algorithms, small lookahead values will usually result in very limited concurrent execution unless other techniques (e.g., optimistic event processing) are used.

The logical time creep problem is an RTI implementation issue and is not discussed further here. Version 0.X of the RTI used the Chandy/Misra/Bryant algorithm, and was thus susceptible to this problem. This problem has subsequently been addressed by the familiarization version of the RTI, F.0. Further experimentation is required to determine if adequate performance has been achieved in this version.

The remainder of this document addresses the first three requirements. Changes to the time management services are proposed that satisfy all three requirements.

4.2 Repeatability with Non-Zero Lookahead

As mentioned earlier, the RTI can facilitate, but cannot *guarantee* repeatability. A number of other issues must be addressed such as ensuring the same random number generator seeds are used, the same inputs and initial state are used, floating point round off is the same from one execution to the next, etc. The RTI supports repeatability to the extent that it enables repeatable ordering of simultaneous events.

Requirements 1 and 2 (repeatability and federate control of the ordering of simultaneous events) can be achieved with the current specification of the time management services assuming non-zero lookahead. The following property in the current specification can be exploited to satisfy these two requirements:

> **Property A:** if the RTI issues a **Time Advance Grant** to time T, the RTI guarantees that all messages with time stamp equal to T (or less than T) have been delivered to the federate.

This implies that all simultaneous events with time stamp T have been passed to the federate. The federate may simply order these events according to whatever criteria is appropriate, and process the simultaneous events in that order. So long as the federate uses a repeatable algorithm for ordering simultaneous events, both requirements 1 and 2 will be satisfied.

It is noteworthy that the above approach does *not* require the RTI to deliver simultaneous events to the federate in a repeatable fashion, i.e., messages for a set of simultaneous events may be delivered by the RTI in a different order from one execution to the next. Because the federate will order events according to its own criteria, it does not matter what order the RTI delivers them. The current specification of time management services actually calls for the same order of delivery, though this functionality has not been implemented in either Version 0.X or F.0 of the RTI. The fact that federates must be able to explicitly control the order of simultaneous events suggests that this functionality is not required. However, RTI automated ordering of simultaneous events does provide a simple means for achieving repeatable executions where arbitrary ordering of simultaneous events is acceptable because the federate can simply process the events in the order that they are delivered by the RTI.

4.3 Repeatability with Zero Lookahead

The approach described above for non-zero lookahead fails when zero lookahead is allowed because it is impossible to maintain property A (delivery of all events with time stamp T when the RTI issues a grant to time T). This is true in both sequential and distributed simulations. A proof of this statement follows.

Theorem 1: If zero lookahead is allowed, it is impossible for a **Time Advance Grant** to time T to guarantee it has delivered all messages with time stamp equal to T.

> ***Proof* (by contradiction):** Suppose an RTI claimed to support both zero lookahead and Property A. Consider the following scenario: (a) Federate F_1 invokes **Next Event Request**, and receives a **Time Advance Grant** to time T. By property A, the RTI guarantees it has delivered all messages with time stamp T (or less) to federate F_1. (b) Federate F_1 performs an **Update Attribute Values** with time stamp T. It can do this because zero lookahead is allowed. (c) Federate F_2 receives a message for this attribute update with time stamp T, and generates a second **Update Attribute Values** with time stamp T. (d) The second update generates a new message that is sent to federate F_1, with time stamp equal to T. This last message contradicts the original assertion that Federate A had received all messages with time stamp T.

Thus, there is a natural tension between zero lookahead and federate controlled ordering of simultaneous events. The latter requires that the RTI guarantee the federate that it has delivered *all* messages with time stamp T so that the federate can produce an appropriate, repeatable ordering of the simultaneous events (otherwise the federate must determine what other simultaneous events it might receive in the future, a difficult task), but the above discussion indicates that this is impossible if zero lookahead is allowed. The proposed solution to this dilemma is to (1) allow zero lookahead federates, but (2) provide a separate mechanism where a federate wishing to receive all events at time T can do so, but at the cost that it must *temporarily* have a non-zero lookahead in order to allow such a mechanism to be implemented.

Two "flavors" of the primitives for advancing logical time are proposed. For the purposes of discussion, changes to the **Next Event Request** service are described, however, identical changes are also proposed for the **Time Advance Request** service. The first flavor is a new service called **Next Event Request Available** that is similar to the **Next Event Request** service now provided in the HLA, but changes the semantics of the resulting **Time Advance Grant** to accommodate zero lookahead federates. The second flavor is essentially the same as the current service, and accommodates federates needing to have all simultaneous events delivered to it when the grant is issued.

Specifically, to allow zero lookahead federates, Property A must be replaced with a weaker property. Property B, defined below, is defined to accommodate zero lookahead.

> **Property B:** if the RTI issues a **Time Advance Grant** to time T in response to

> an invocation of the **Next Event Request Available (T')** service, the RTI guarantees that all messages with time stamp *strictly less than* T have been delivered to the federate. Some messages with time stamp T may have been delivered, however, additional messages with time stamp equal to T may also be delivered later in the execution.

Second, to accommodate federate controlled ordering of simultaneous events, Property C is proposed.

> **Property C:** if the RTI issues a **Time Advance Grant** to time T in response to an invocation of the **Next Event Request (T')** service, the RTI guarantees that all messages with time stamp less than *or equal to* T have been delivered to the federate. The federate is prohibited from generating any new events with time stamp less than *or equal to* T subsequent to invoking the **Next Event Request (T')** service.

In property C, delivery of *all* events with time stamp of T (or less) enables a federate to collect all simultaneous events at time T, and order them according to federation-defined rules. The RTI thus provides the necessary support for federation controlled, repeatable executions. The second part of property C prevents the federate from generating new events with time stamp of T, and is necessary to realize the guarantee that all events with time stamp T have been delivered. In effect, Property C forces the federate to (temporarily) have a non-zero lookahead. Prior to invoking the **Next Event Request (T')** service that resulted in the issue of the **Time Advance Grant** (based on property C), the federate is allowed to generate zero lookahead events. After the federate invokes **Next Event Request (T')** and receives a **Time Advance Grant** to advance to logical time T, it may resume scheduling zero lookahead events once the federate advances beyond logical time T by invoking the **Next Event Request Available** service.

Different federates within a single execution may use the different flavors of **Next Event Request**. Further, a *single* federate may use both of these two services, as illustrated in the example that follows. Federates using the existing time management mechanisms require no changes to utilize the new services.

5. EXAMPLE

A federate with zero lookahead requiring repeatable executions and federate control of the ordering of simultaneous events might use these primitives as shown in Figure 1. In this example, the first invocation of **Next Event Request Available** is used to advance the federate to the time of the next event. Subsequent invocations of **Next Event Request Available** can be used to receive additional messages, e.g., responses to queries, without advancing logical time. The final invocation to **Next Event Request** is used to flush out any remaining events containing the same time stamp. All events that are generated with time stamp equal to the current time (zero lookahead) must be generated prior to this latter invocation of **Next Event Request(T).**

6. IMPLEMENTATION

Implementation of this mechanism within the RTI is relatively straight-forward. Invocation of **Next Event Request** (or **Time Advance Request**) is essentially identical to the implementation of the service that is defined now, except the RTI must simultaneously set the lookahead of the federate to *epsilon* (the smallest time stamp increase possible) if the federate's lookahead is currently zero. This change is erased once the federate advances to a new logical time.

After the **Next Event Request Available (Time Advance Request Available)** service is invoked, the RTI issues a **Time Advance Grant** to the federate once its internally computed LBTS is greater than *or equal to* the time of the grant (the current definition requires the LBTS to *exceed* the time of the grant). If the time specified by the **Next Event Request Available** (or **Time Advance Request Available**) is the same as the federate's current logical time, the RTI might immediately return a **Time Advance Grant** after delivering any relevant messages, or it may delay issuing the grant until it has either (1) delivered at least one message to the federate, or (2) some federate-specified time out has expired.

An important concern with the inclusion of zero lookahead federates is deadlock. Deadlock results if all of the federates are blocked on a request to advance logical time, but the RTI cannot issue a grant to any federate. In the current time management specification, this will occur if zero lookahead is introduced without any additional changes to the existing time management services. The mechanism proposed here is not prone to deadlock. To see this, suppose all of the federates are blocked on a request to advance logical time. Each such request includes a time parameter. Let T be the value of the smallest time parameter specified in any request. Consider the set of federates that have specified T in their request (in general, there may be more than one). If at least one of these federates is blocked on a **Next Event Request Available**, the RTI can issue a grant to those federate(s) to proceed because the RTI need only guarantee that no additional messages with time strictly less than T will later be received. This cannot occur (even with zero lookahead) because all federates are ready to advance to T (at least). If all of the federates attempting to advance to time T are blocked on a **Next Event Request** call, Property C guarantees (in effect) a nonzero lookahead, so all of these federates can be issued a grant, again breaking the deadlock. Thus in either case, no deadlock situation can occur.

```
/* now is a local variable tracking the logical time of the federate */
while (simulation still in progress)
    Determine time stamp of next local event notice, TSlocal is time stamp of this notice

    /* advance logical time to time of next event */
    invoke Next Event Request Available (TSlocal) service
        Receive messages (via Reflect Attribute Values/Receive Interaction services)
    honor RTI service request for Time Advance Grant
    now = time of grant

    /* perform any zero lookahead operations, e.g., queries, at the current time */
    invoke any Update Attribute Values/Send Interaction services with zero lookahead

    /*
     * The federate may now repeat calls to Next Event Request Available (now)
     * and Update Attribute Values/Send Interaction to exchange additional messages
     * without advancing logical time.  For example, additional queries may be issued here.
     */

    /* retrieve all remaining events at the current time; logical time does not advance here */
    invoke (Next Event Request (now) service
        Receive messages (via Reflect Attribute Values/Receive Interaction services)
    honor RTI service request for Time Advance Grant

    /* order any simultaneous events, process them in proper sequence */
    sort local and external events received above using federate defined ordering rules
    process local/external event notices in order, providing any changed information
        (new attribute values or interactions) to the RTI via the Update Attribute Values
        and/or Send Interaction services.
```

Figure 1. Example illustrating a federate with zero lookahead and requiring repeatable executions and federate controlled ordering of simultaneous events.

7. PROPOSAL

The following changes to the HLA time management services are proposed. This proposal satisfies requirements 1 (repeatability), 2 (federation control of simultaneous events), and 3 (zero lookahead).

Zero lookahead is allowed. Two new services called **Next Event Request Available** and **Time Advance Request Available** are defined, and the existing **Next Event Request** and **Time Advance Request** services are modified:

- **Next Event Request/Time Advance Request (T'):** a **Time Advance Grant** to logical time T issued in response to this request indicates all events with time stamp less than or equal to T have been delivered to the federate, and the federate may *not* generate any new events with time stamp equal to T, even if the federate's lookahead is zero. This is essentially the same as the time management services that are now defined. The federate may resume scheduling events with zero lookahead once the federate's logical time has advanced beyond T.
- **Next Event Request Available/Time Advance Request Available (T'):** a **Time Advance Grant** to logical time T issued in response to this request indicates all events with time stamp less than T have been delivered to the federate, and those buffered in the RTI with time stamp T have been passed to the federate, but additional events may be later delivered with time stamp equal to T. The federate may generate additional events with time stamp equal to T if its lookahead is zero.

In order to support repeatability when the arbitrary ordering of simultaneous events is satisfactory to the application, the RTI continues to be required to deliver messages for simultaneous events to the federate in the same order from one execution to the next.

8. ACKNOWLEDGMENTS

This paper is the result of extensive discussions in the HLA Time Management working group. In particular, Emmet Beeker, John Chludzinski, Dannie Cutts, Jim Gump, John Hancock, Frank Hodum, Steve Jackson, Andreas Keurkes, Reed Little, Ray Mandery, Larry Mellon, Katherine Morse, Michael Petty, Kent Pickett, Jonathan Prescott, Paul Reynolds, Ed Roberts, Sudhir Srinivasan, Jeffery Steinman, Dan Van Hook, Richard Weatherly, Darrin West, Douglas Wood, and Phil Zimmerman contributed to discussions concerning this issue.

9. REFERENCES

[1] Defense Modeling and Simulation Organization, "HLA Interface Specification, V. 1.0," U.S. Department of Defense, Washington D.C. August 1996.

[2] Defense Modeling and Simulation Organization, "HLA Time Management Design Document, V. 1.0," U.S. Department of Defense, Washington D.C. August 1996.

[3] H. Mehl, "A Deterministic Tie-Breaking Scheme for Sequential and Distributed Simulation," in *Proceedings of the Workshop on Parallel and Distributed Simulation*, vol. 24, M. Abrams and P. Reynolds, Jr., Eds.: Society for Computer Simulation, 1992, pp. 199-200.

[4] P. Reiher, F. Wieland, and P. Hontalas, "Providing Determinism in the Time Warp Operating System -Costs, Benefits, and Implications," in *Proceedings of the Workshop on Experimental Distributed Systems*. Huntsville, Alabama: IEEE, 1990, pp. 113-118.

[5] D. West, L. Mellon, J. Ramsey, J. Cleary, and J. Hofmann, "Infrastructure for Rapid Execution of Strike-Planning Systems," in *Proceedings of the 1995 Winter Simulation Conference*. Crystal City, Virginia, 1995, pp. 1207-1214.

[6] T. J. Schriber, *Simulation Using GPSS*. New York: John Wiley & Sons, 1974.

[7] L. Schruben, "Simulation Modeling with Event Graphs," *Communications of the ACM*, vol. 26, 1983.

[8] T. Nakanishi, "Modeling Problems in the Processing of Simultaneous Events," in *Proceedings of the Summer Computer Simulation Conference*: Society for Computer Simulation, 1992.

[9] F. Wieland, "The Threshold of Event Simultaneity," MITRE, Corp. 1996.

[10] K. M. Chandy and J. Misra, "Distributed Simulation: A Case Study in Design and

Verification of Distributed Programs," *IEEE Transactions on Software Engineering*, vol. SE-5, pp. 440-452, 1978.

[11] K. M. Chandy and J. Misra, "Asynchronous Distributed Simulation via a Sequence of Parallel Computations," *Communications of the ACM*, vol. 24, pp. 198-205, 1981.

[12] V. Jha and R. Bagrodia, "Simultaneous Events and Lookahead in Simulation Protocols," University of California, Los Angeles Technical Report 960043, 1996.

[13] R. M. Fujimoto, "Performance Measurements of Distributed Simulation Strategies," *Transactions of the Society for Computer Simulation*, vol. 6, pp. 89-132, 1989.

[14] R. M. Fujimoto, "Parallel Discrete Event Simulation," *Communications of the ACM*, vol. 33, pp. 30-53, 1990.