Time Management in the High Level Architecture

Richard M. Fujimoto
College of Computing
Georgia Institute of Technology
Atlanta, GA 30332-0280
fujimoto@cc.gatech.edu

ABSTRACT

Time management is required in simulations to ensure temporal aspects of the system under investigation are correctly reproduced by the simulation model. This paper describes the time management services that have been defined in the High Level Architecture. The need for time management services is discussed, as well as design rationales that lead to the current definition of the HLA time management services. These services are described, highlighting information that must flow between federates and the Runtime Infrastructure (RTI) software in order to efficiently implement time management algorithms.

1. Introduction

The Defense Modeling and Simulation Office (DMSO), through its High Level Architecture (HLA) initiative, is addressing the continuing need for interoperability between new and existing simulations within the U. S. Department of Defense. The HLA builds upon and generalizes the results of the Distributed Interactive Simulation (DIS) effort (DIS Steering Committee 1994) and related activities such as the Aggregate Level Simulation Protocol (ALSP)(Wilson and Weatherly 1994). The HLA activity began in March 1995. It was approved by the Under Secretary of Defense (Acquisition and Technology) as the common technical architecture for modeling and simulation in the U.S. Department of Defense in September 1996. At the time of this writing the HLA is undergoing standardization through the IEEE.

The HLA consists of (1) the rules that define the basic principles underlying the HLA, (2) the object model template (OMT) that provides a standard format for describing information of common interest to more than one simulator (federate), and (3) the interface specification (I/F Spec) that defines the interface to the Run Time Infrastructure (RTI) that provides the means for simulators to coordinate the execution and exchange of information. An HLA simulation typically consists of a collection of simulators (e.g., vehicle simulators) interconnected through the RTI. Each simulator is referred to as a *federate*. The collection of federates interacting through the RTI is referred to as a *federation*.

The RTI can be viewed as a special purpose distributed operating system that provides a set of services used to interconnect simulations. The I/F Spec defines a set of services

invoked by simulations or by the Run-Time Infrastructure during a federation execution. HLA runtime services fall into the following categories:

- *Federation management.* This includes services to create and delete federation executions, to allow simulations to join or resign from existing federations, and to pause, checkpoint, and resume an execution.
- *Declaration management.* These services provide the means for simulations to establish their intent to publish object attributes and interactions, and to subscribe to updates and interactions produced by other simulations.
- *Object management.* These services allow simulations to create and delete object instances, and to produce and receive individual attribute updates and interactions.
- *Ownership management.* These services enable the transfer of ownership of object attributes (and the right to modify the value of these attributes) during the federation execution.
- *Time management.* These services coordinate the advancement of logical time, and its relationship to wallclock time during the federation execution.
- *Data distribution management.* These services allow large federations to improve the efficiency of the data distribution mechanisms by reducing the amount of data that is sent between federates.

This article is concerned with the time management services and message ordering aspects of the object management services. Fundamental concepts and design rationales underlying the time management services are first discussed. This is followed by a discussion of principal time management services from the perspective of a federate, focusing on how one would use them. Finally, additional aspects of the time management services are discussed, focusing on the information and functionality that must be provided in the interface to the RTI to realize an efficient implementation of time management algorithms, or to support certain capabilities such as optimistic event processing.

2. Time

One of the most common points of confusion when discussing time management concerns what is meant by *time*. This is because there are three different types of time that are important in distributed simulations:

- *Physical time* refers to time in the physical system, i.e., the system being modeled by the simulation. For example, in a simulation of the attack on Pearl Harbor, physical time might extend from midnight until 6 o'clock in the afternoon on December 7, 1941.
- *Simulation time* refers to the simulator's representation of time. In the Pearl Harbor simulation, simulation time might be represented as a double precision floating point value that can hold values in the interval [0.0, 18.0] where a unit of simulation time corresponds to an hour of physical time.
- *Wallclock time* refers to time when the simulator is executed. For example, the Pearl Harbor simulator might require three and a half hours to execute. If it were executed

in the afternoon of September 10, 1996, wallclock time might extends from 1330 until 1700 on that day.

It is important to keep these different notions of time distinct when discussing time management. Here, we are primarily concerned with simulation time. For example, later event time stamps will be discussed. These time stamps refer to points in simulation time. In the HLA, simulation time is represented as points along a global *federation time axis.* At any instant during the execution, a federate will be at a specific point along this axis referred to as its *federate time.* For example, a federate time of 4.0, might indicate that federate has simulated the first four hours of the battle.

In some cases, advances in simulation time are paced to be in synchrony with advances in wallclock time. For example, a training simulation where humans are embedded into a computer generated virtual environment must be paced so that the environment appears realistic to the trainees. Simulations designed to operate in this fashion are often called *real-time simulations.* A variation on this approach are so-called *scaled real-time simulations* where simulation time is paced to run at some agreed upon factor relative to wallclock time, e.g., the simulator may run twice the speed of wallclock time. In either case, durations in simulation time have a linear relationship with durations in wallclock time, i.e.,

$$\Delta T = S * \Delta W$$

where ΔW is a duration in wallclock time, S is a scale factor, and ΔT is the corresponding duration in simulation time.

On the other hand, in *as-fast-as-possible* simulations one attempts to complete the simulation as quickly as possible, so execution is *not* paced to have a direct relationship to wallclock time. For example, the simulator may run faster-than-wallclock time early in the execution, and slower-than-wallclock time during other parts of the execution. Analytic wargame simulations often fall into this category. The HLA is intended to support both real-time and as-fast-as-possible simulators and simulation executions.

3. Why do we need time management?

At first glance, it may not be clear why there is a need for time management. Why not just "hook together the simulators" and have them send messages to each other whenever one simulator performs some action of interest to another. This can lead to problems as will be seen momentarily.

Consider a real-time simulation consisting of three simulators (federates): a federate modeling a tank, one modeling a target, and a third modeling observers. Suppose the tank federate fires upon the target and destroys it, while the observers monitor this exchange. Figure 1 illustrates the execution of the federation during this interchange. The tank federate first generates a message indicating it has fired. This message is received by the target federate that models the target being destroyed and generates a

message indicating this fact. The observer federate receives both messages, but the message containing the tank firing event has been delayed in the network, causing the target destroyed event to be received first. Thus, the observer sees the target being destroyed before it has been fired upon!

The basic problem stems from the fact that the simulated world may not correctly reproduce temporal aspects of the real world that is being modeling. For example, delays in the real world depend on quantities such as the speed of light. In the simulated world, delays depend upon quantities having nothing to do with the simulation model, e.g., the delay encountered by a message as it travels through the network. This can lead to anomalies such as the cause appearing to happen *after* the effect.

Further, if no precautions are taken, it is possible different simulators will receive messages for the same set of events in different orders. If a second observer is added to the federation depicted in Figure 1, it may receive messages for the fire and target destroyed events in the correct order. If the behavior of each simulator depends on the ordering of events this can lead to other inconsistencies in the simulation execution.

Another problem is that repeated executions of the simulation with the same initial state and external inputs may produce entirely different results. One reason for this is the order that messages are delivered to each federate depends on properties such as communication delays in the network that may change from one execution to the next. It is sometimes important that repeated executions of the simulation yield the same results. For example, in defense simulations, the results produced by the simulator may be used to make acquisition decisions. Simulation results may be audited by the General Accounting Office, which may require the simulations to be repeated, e.g., to verify their authenticity.
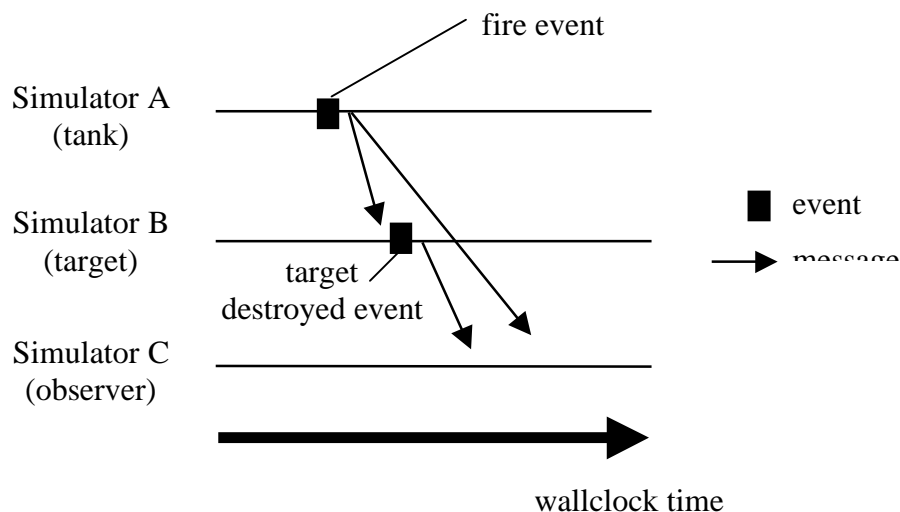

Figure 1 Scenario illustrating the need for time management. The observer simulator receives the "target destroyed" message before the "fire" message.

The importance of correctly reproducing temporal relationships in the simulation model depends on the simulation application. This is often very important in analytic simulations because impossible event ordering may not have been anticipated by the model developers, and could cause a simulator to fail. Further, variations in event timings because of network latency may introduce bias in the statistics computed by the simulation. On the other hand, in a training application non-causal ordering of events such as that shown in Figure 1 may not be perceptible to human participants if they occur in rapid succession. Even if they are perceptible, they may not compromise the training objectives of the exercise, or they may happen so infrequently that they can be tolerated. Moreover, if the computational overheads necessary to prevent such anomalies preclude real-time performance, there may be little alternative other than to accept that they may occur and design around them.

In the HLA, as is the case in most simulation applications, temporal anomalies are eliminated by assigning a time stamp (in simulation time) to each event, and ensuring that events are delivered to the federate in time stamp order. Because the "cause" will always be assigned a smaller time stamp than any "effect", this guarantees that causal relationships such as that shown in Figure 1 will be correctly reproduced by the simulation model. In particular, the RTI will delay delivery of the target-destroyed event until after the fire event (that has a smaller time stamp) has been delivered. In addition, the time management services ensure a federate will not receive an event in its past, i.e., an event with time stamp less than its current federate time. Both time stamp order and the constraint that no events are delivered in a federate's past can be relaxed by federates not requiring this capability.

4. Design Rationale

A key goal of the HLA is to foster interoperability and reuse of simulations. To facilitate reuse, the time management services must be sufficiently flexible to accommodate the wide variety of internal time management mechanisms commonly used in defense simulations today. To support interoperability, the time management services must allow simulations using different internal time management mechanisms to be combined in a single federation execution.

An important design principal that is used to foster interoperability is *time management transparency.* This means the local time management mechanism used within each federate must *not* be visible to other federates. For example, a federate using an event oriented time flow mechanism need not know the federate with which it is interacting is using an event-oriented or a time stepped mechanism. Federates do not explicitly indicate to the RTI the local time management mechanism being used.

The principal types of time management mechanisms used for defense simulations are enumerated below. These cover a broad spectrum of applications including analysis, training, and test and evaluation (T&E). The requirements of these simulations drove the design of the HLA time management services.

- *Event driven.* The federate processes local events and those generated by other federates in time stamp order; federate time typically advances to the time stamp of each event as it is processed. Process-oriented simulation constructs that are often found in modern simulation languages are included in this category because they are typically built on top of an event execution mechanism.
- *Time stepped.* Each time advance made by the federate is of some fixed duration of simulation time, called a time step. The simulator does not advance to the next time step until all simulation activities associated with the current time step have been completed. Activity scanning, another time flow mechanism sometimes used in discrete event simulation, is essentially a variation on the time stepped mechanism.
- *Parallel discrete event simulation.* Federates executing on multiprocessor systems must be synchronized internally using a conservative or an optimistic synchronization protocol. In a conservative protocol, each *logical process* within the federate must process its events in time stamp order. Logical processes may have to wait until this condition can be guaranteed. Optimistic synchronization protocols allow logical processes to process events out of time stamp order, but provide a means to recover from such errors, typically through the use of a rollback mechanism.
- *Wallclock time driven.* Here we are referring to simulators that derive their current simulation time directly from wallclock time, as was described earlier. This is in contract to simulators using one of the other time flow mechanisms described above that pace their execution to advance in synchrony with wallclock time. Wallclock time driven federates do not necessarily require that events be processed in time stamp order. These simulations usually have hard and/or soft real-time constraints so that interactions with humans and/or physical devices occur in a timely fashion.

Time management in any federation must be realized jointly by the federates and the RTI. Thus a key question is: What functionality should be implemented within the RTI, and what should be realized within individual federates?

Because event driven federates must process both internal events and those generated by other federates in time stamp order, the federate must, in effect, merge the stream of events generated by other federates with local events to produce one stream of time stamp ordered events. Further, precautions must be taken to ensure the federate does not receive events in its past, i.e., events with time stamp less than the simulator's current federate time. These tasks are accomplished in the HLA time management services by:
- a time stamp order (TSO) message delivery service, and
- a protocol where federates explicitly request advances in their federate time, and the RTI provides a grant of this request when it can guarantee no events will later arrive containing a smaller time stamp.

Time stepped federates must be able to determine when all events in the current time step that are produced by other federates have been received. Only then can it complete the computation for the current time step, and advance to the next time step. This requirement is satisfied in the HLA with a variation on the protocol described above for advancing federate time.

Parallel discrete event simulations using conservative synchronization have similar requirements as event driven federates. A modest amount of additional functionality is required to support parallel simulators using optimistic time management. More will be said about this later.

Time management based on wallclock time is commonly used in virtual simulators used for training (e.g., vehicle simulators) and hardware-in-the-loop T&E applications. These simulators typically use an entirely different approach to time management than that described above. Commonly used mechanisms include:

- *Clock synchronization* algorithms are required to ensure that hardware clocks generating wallclock time values in different, possibly geographically distributed processors are properly synchronized. This is needed because hardware clocks in different computers drift relative to each other. This can lead to serious problems in the distributed simulation if the differences become large.
- *Real*-time *scheduling* is often used to guarantee that computations are completed by certain deadlines in order to ensure timely interactions with hardware devices and/or human participants. Task deadlines and laxities are often used to schedule the computations so that deadlines can be met.
- *Time compensation* is used to take into account communication latency. For example, suppose a simulator for a moving vehicle generates a message indicating its current position, speed, and direction of motion. If the message is received 100 milliseconds later, the destination can extrapolate the current position of the vehicle using dead reckoning techniques in order to estimate the vehicle's position when the message was received.

In the HLA, these mechanisms must be implemented within the federate. Unlike message ordering and time stamping requirements that are largely independent of the goals of the federation and what is being simulated, specifics concerning the above mechanism are highly dependent on the federation objectives and details of the models. For example, hardware clocks must often be more closely synchronized in hardware-in-the-loop T&E applications compared to interactive training simulations. Scheduling algorithms require detailed information concerning the computations performed within the federate, and thus are not well suited for implementation within the RTI. Time compensation techniques require information concerning the semantics of what is being simulated. Such information is not available within the RTI.

For this reason, the HLA time management services provide a minimalist approach to addressing time management concerns for these applications. The principal requirements for these applications with respect to the RTI are (1) minimal, predictable communication latency, and (2) minimal computational and communication overhead imposed by the RTI. Toward this end, the HLA provide basic communication services where messages are delivered with minimal latency, and little additional functionality for time management. In particular, the overheads associated with realizing time stamp order and control over time advances are eliminated if these services are not used.

Here, federate time refers to the current simulation time of a federate. For federates that can send or receive TSO messages, federate time is referred to as *logical time*, to distinguish these federates from those using a federate time derived directly from wallclock time. At any instant during the execution of a federation, different federates may, and often do, have different logical time values.
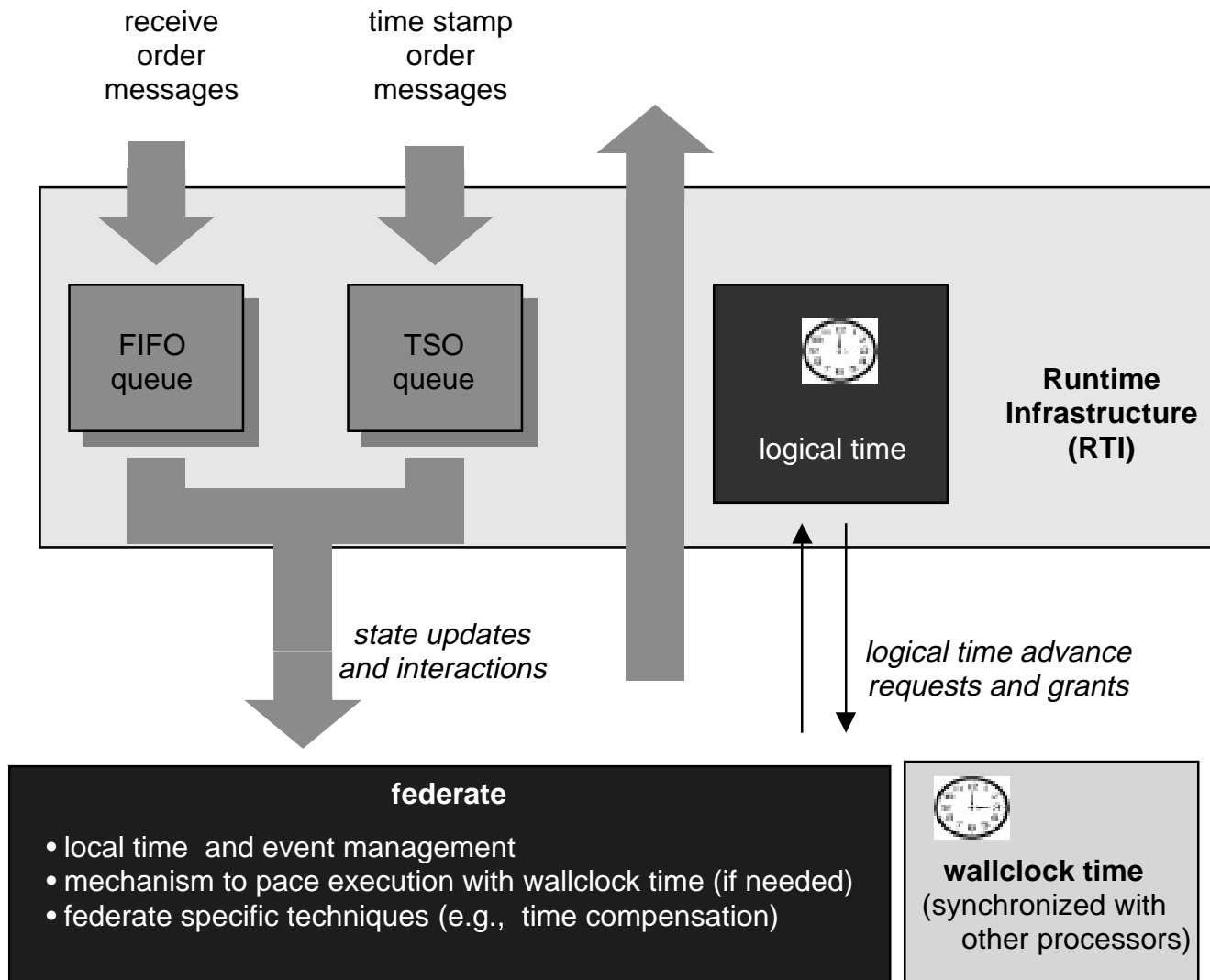

Figure 2. Logical view of time management in the HLA.

5. Time Management Services

A logical view of time management component of the HLA is illustrated in

Figure 2. The services include two key components: a time stamp order delivery service, and a protocol for simulations to advance their logical time.

5.1 Message Order and Time Stamps

Incoming messages are categorized as either receive order (RO) or time stamp ordered (TSO). Receive ordered messages are simply placed in a queue when they arrive, and are immediately eligible for delivery to the federate. The ordering of these messages is arbitrary. TSO messages are assigned a time stamp by the federate generating the message, and are delivered to each receiving federate in order of non-decreasing time stamps. Incoming TSO messages are placed into a queue within the RTI, but are *not* eligible for delivery until the RTI can guarantee there are no other TSO messages destined for that federate containing a smaller time stamp, *and no others will later arrive from other federates.* To ensure the latter property, the RTI must compute a lower bound on the time stamp of future messages it may receive from other federates. More will be said about this later. Because the RTI may not be able to deliver TSO messages immediately upon receipt, TSO messages may incur a higher message latency than RO messages. Federates may send and receive both TSO and RO messages.

Messages containing identical time stamps, referred to as *simultaneous events*, are delivered to the federate in an arbitrary order. Often it is important to order these messages in a particular fashion. To address this issue, the federate receiving the messages can buffer and order the messages itself. To support this, the RTI provides a means for specifying when the federate has received all simultaneous events with a given time stamp value. Alternatively, federates may include tie-breaking fields in the message time stamp to order simultaneous events. This requires federation specification of the format and meaning of the time stamp field and logical time values and duration. To support this, time values can be specified by the federation using an abstract data type. This allows the time representation to be tailored to meet the requirements of different federations. In addition to specifying the time stamp format, which must be documented in the Object Model Template, the federation must also specify certain operators such as comparison of time stamp values. This capability enables HLA federations to utilize techniques such as those described in (Mehl 1992) to deterministically order events, e.g., to facilitate repeatable executions.

Receive ordered messages may also, optionally, contain a time stamp value. In this case, the time stamp value is simply passed, uninterpreted by the RTI, to the destination federate.[1]

5.2 Advancing Logical Time

The second portion of the time management component of the HLA provides a mechanism to advance simulation time within each federate. As mentioned earlier, the RTI guarantees a federate will not receive any TSO message with time stamp less than its current logical time. To realize this capability, federates cannot autonomously advance

---

[1] Specification of the representation of time as an abstract data type and allowing time stamps on receive ordered messages were added to the interface specification in late 1998.

their logical time. Rather, federates must explicitly request that their logical time be advanced, and advances do not take place until the RTI explicitly grants them. This protocol for advancing logical time is central to the HLA time management services.

A time management "cycle" consists of three steps. First, the federate invokes a time management service to request its logical time to advance. Next, the RTI delivers some number (possibly zero) of messages to the federate. A message is delivered to the federate by the RTI invoking a federate defined procedure, e.g., **Reflect Attribute Values** to deliver new values for object attributes, or **Receive Interactions** to deliver interaction events. The RTI completes the cycle by invoking a federate defined procedure called **Time Advance Grant** to indicate the federate's logical time has been advanced.

The two principal mechanisms for a federate to request its logical time to be advanced are the **Time Advance Request (TAR)** and **Next Event Request (NER)** services. TAR is well suited for federates that internally use a time stepped mechanism, and NER is the preferred primitive for event driven federates. There is no restriction concerning what federates can invoke which primitive, however. Any federate can exclusively use NER or TAR, or it can intermix calls between the two. These two time management services are defined as follows:

- **Time Advance Request (T):** The federate invokes this service to request its logical time to be advanced to T. All RO messages in the RTI's internal queues, and all TSO messages with time stamp less than or equal to T are delivered to the federate after this service has been invoked. When no additional TSO messages with time stamp less than or equal to T are forthcoming (or will be generated by another federate in the future), the RTI calls the federate's **Time Advance Grant** procedure with parameter T to indicate the federate's logical time has been advanced to T.
- **Next Event Request (T):** An event driven federate will typically invoke this service when it has completed all simulation activity at the current logical time, and is ready to advance to a new time. The parameter T specified in the NER request indicates the logical time to which the federate would like to advance, if there are no other events from other federates containing a smaller time stamp. Typically, T is the time stamp of the next event in the federate's local set of pending events. After invoking this service, the RTI will deliver all RO messages in its internal queue. If there are no TSO messages with time stamp less than or equal to T, and none will be received in the future, the RTI invokes the federates **Time Advance Grant** procedure indicating its logical time has been advanced to T. Otherwise, the RTI will deliver the next smallest TSO message destined for the federate (with time stamp T' where T' ≤ T) and all other messages with time stamp T'. The RTI then calls **Time Advance Grant** with parameter T'. In this case, the federate's logical time is advanced to T'.

Now = 0; /* current time */	**Federate**	**RTI**
while (simulation not completed)	Main: …	
T=time of next event in PES	T=time of next event in PES	
NextEventRequest(T)	Main: NextEventRequest(T)	

Column (a):

```
PendingNER = TRUE;
while (PendingNER) Tick();
Now = time of next event in PES
Process events in PES at time Now
End-while


ReflectAttributeValues()
  Store event into PES


TimeAdvanceGrant()
  PendingNER = FALSE;
```

(a)

Column (b):

```
Main: PendingNER = TRUE;          NER: … return
Main: Tick();
                                  Tick: …

                                  Tick: ReflectAttributeValues

RAV: store event into PES
RAV: return
                                  Tick: return
Main: Tick();
                                  Tick: …
                                  Tick: TimeAdvanceGrant
TAG: PendingNER=FALSE
TAG: return
                                  Tick: return
Main: Now = time of next event
Main: Process events in PES
```
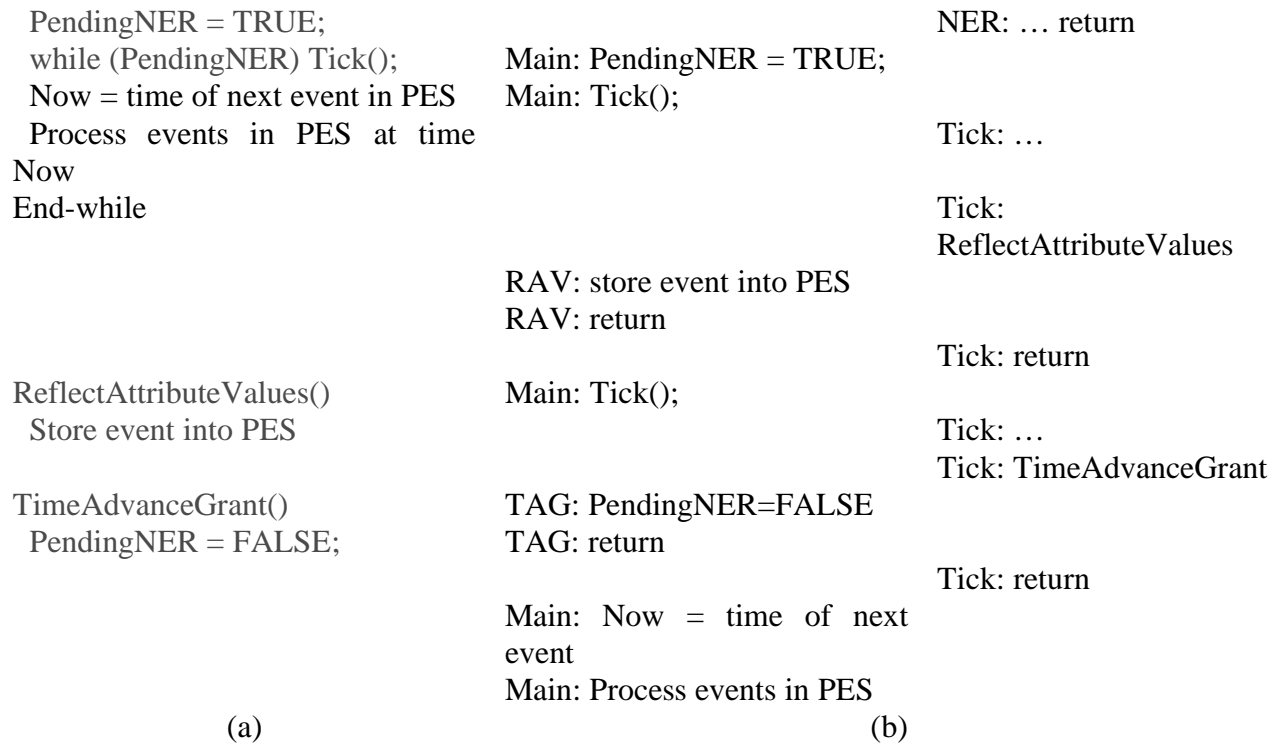
(b)

Figure 3. Example for an event driven federate. (a) sketch of code example. PES denotes the internal pending event set of the federate. The black portion represents code that exists in the original simulator. The colored code represents portions added to interface the simulator to the RTI. (b) sample execution, where wallclock time advances from the top to bottom of the figure. RAV denotes code in ReflectAttributeValues, TAG denotes code in TimeAdvanceGrant, and Main denotes code in the main federate event scheduling loop shown on (a).

5.3 Example

The code for an event driven federate is sketched in Figure 3(a). This example assumes a *single threaded* implementation where the RTI is realized as a library that is compiled into the code for the federate. Many existing implementations of the RTI operate in this fashion. With this approach, a key question concerns how execution is scheduled between the RTI and the federate. This problem is solved by an RTI procedure called Tick that is called by the federate to turn the CPU over to the RTI to allow it to perform its operations. Specifically, delivery of messages and time advance grants are implemented by calls from the RTI to the federate within the Tick procedure.

Figure 3(b) illustrates the execution sequence for a time advance cycle for the code shown in Figure 3(a). The federate first invokes the RTI's NextEventRequest (NER) service to request delivery of TSO messages, or to advance its logical time to the time stamp of its next local event. After NER has been called, the federate sets a flag indicating it is waiting for an NER cycle to complete, and calls tick to turn the CPU over to the RTI. In this example, there is one event produced by another federate that has

time stamp less than the next event in the federate's local pending event set (PES). The Tick procedure calls the federate's ReflectAttributeValues procedure to deliver this event to the federate. Here, the event is simply placed into the PES along with other, locally generated events. After this has been done, Tick returns control to the federate. Because the PendingNER flag is still TRUE, the federate will again call Tick to request additional events, or a time advance grant if there are none. This time, the RTI determines there are no more events with the same time stamp as the event it just delivered, and none are forthcoming from other federates in the future. The RTI issues the grant by calling the federate's TimeAdvanceGrant procedure. This procedure sets the federate's PendingNER flag to FALSE and returns. Tick returns, and the federate now advances its local clock (the Now variable), processes its events, and repeats this cycle.

6. Interface Issues

Time stamp ordering and primitives to advance time form the central components of the HLA time management architecture. The interface is designed so that the RTI is provided with essential information necessary to realize efficient implementations of the time management primitives, but at the same time, RTI implementers are not constrained to use a particular synchronization protocol. In particular, the interface represents a kind of contract between the federate and RTI concerning the time stamp of messages the federate might generate later in the execution. Additional primitives are provided to support federates executing on parallel processors using optimistic synchronization protocols. These aspects of the time management interface are discussed next.

6.1 LBTS Computation

The key to implementing the time management services within the RTI is to compute a quantity called the lower bound on time stamp (LBTS) for each federate. $LBTS_i$, the LBTS value computed for federate i, is a lower bound on the time stamp of messages that may be received that are destined for that federate later in the execution. Two key tasks the RTI must perform are (1) ensure TSO messages are delivered to the federate in time stamp order, and (2) ensure no message is delivered to the federate with time stamp smaller than its current logical time. Once $LBTS_i$ has been computed, the RTI can deliver all TSO messages containing a time stamp less than $LBTS_i$. Further, if the RTI prevents the federate from advancing its logical time beyond $LBTS_i$, then it can guarantee that federate will not receive any messages in its past.
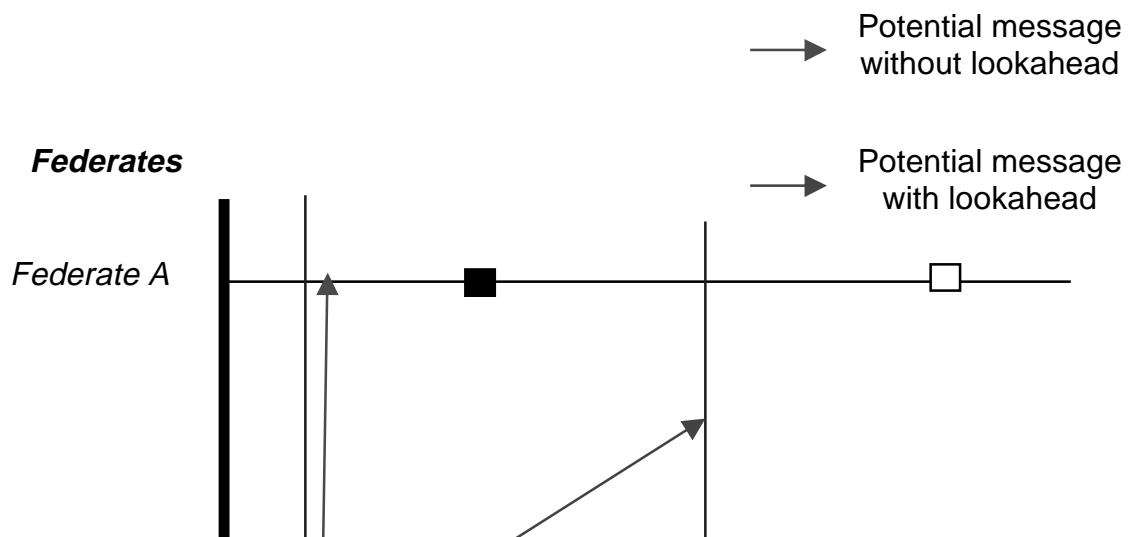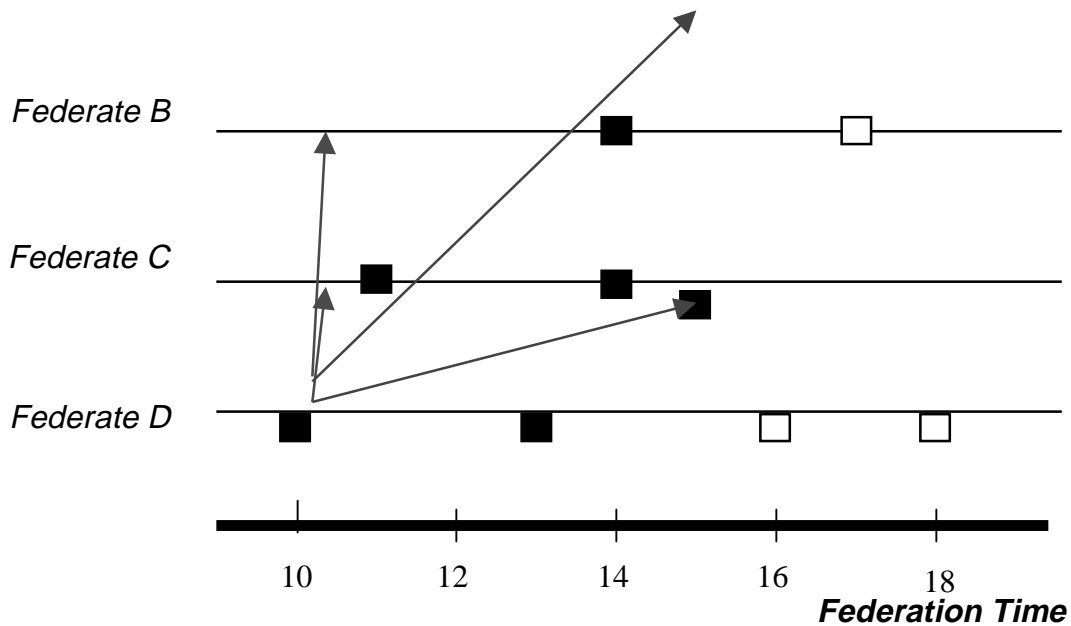
Figure 4. Snapshot of a federation showing unprocessed events and their time stamps. The smallest time stamped event at federation time 10 is the only event the RTI can deliver if there is no lookahead. If each federate is at time 10, and has a lookahead of 5, all of the darkened events with time stamp 15 or less can be delivered.

For example, Figure 4 shows a snapshot of a federation execution containing four federates all at time 10. Each box represents an event, and the (X,Y) coordinate position of the event indicates the time stamp, and destination federate. It is seen that two events, containing time stamps 14 and 17 have been sent to federate B. Suppose $LBTS_B$ has been computed to be 15. This means the time stamp 14 event can be delivered to the federate, but the time stamp 17 event cannot, since it is still possible federate B will later receive another event with time stamp less than 17. Similarly, the RTI cannot advance the logical time of federate B beyond 15, because if it did so, that would leave the possibility the federate could receive a TSO message in its past.

To compute LBTS, the RTI must consider
- the smallest time stamp of any TSO message any federate might generate in the future; the current logical time of a federate is one bound since no federate can generate a TSO message in its past, and
- the time stamps of messages within the RTI and the interconnection network.

Algorithms for computing LBTS are beyond the scope of the present discussion. Interested readers are referred to (Fujimoto 1999).

6.2 Time Constrained and Time Regulating Federates

An HLA federation may include some federates using the time advance and time stamp order delivery services (e.g., analytic wargaming simulations), as well as others that do not (e.g., DIS style training simulations). In such an environment, the RTI must determine those federates that must participate in the LBTS computation, and those that require the results of the computation.

In the HLA, each federate contains two boolean flags that indicate to the RTI its status with respect to LBTS computations: the *time constrained* and *time regulating* flags. A time regulating federate is one that is able to generate TSO messages, and therefore must be considered when computing LBTS values. A time constrained federate is one that may receive TSO messages, and thus requires the result of LBTS computations. The HLA time management services provide primitives for federates to turn on or off their constrained and regulating flags. If a federate does not have its regulating (constrained) flag set when it sends (receives) a TSO message, the RTI will convert that message to an RO message. Federates with their regulating (constrained) flag set can send (receive) both TSO and RO messages. Because the LBTS computations require a certain amount of computational and communication resources, federates would not normally set these flags unless they had specific requirements to operate using the time management services.

6.3 Lookahead

Consider the federation depicted in Figure 4. Assume all four federates have both their constrained and regulating flags set, i.e., each can send and receive TSO messages. Suppose federate D is at logical time 10, and is processing the smallest time stamped event in the federation. This federate could generate a TSO message with timestamp 10 to each of the other federates. This implies none of the other federates can advance beyond logical time 10, so in the snapshot depicted in Figure 4, the other three federates must remain idle until federate D has advanced. Needless to say, this is a very undesirable situation!

A constraint called *lookahead* provides a solution to this problem. Suppose one were to mandate that no simulation may schedule a new event with timestamp less than the federate's current time plus (say) 5. Then any new event generated by federate D must have a time stamp of at least 15. This, would allow each of the other federates to advance up to logical time 15, i.e., the shaded events in Figure 4 with time stamp less than or equal to 15 can be delivered to the federates and processed.

This value 5 in the above example is referred to as the lookahead for the federate. This is because a federate must be able to "look ahead" 5 time units into the future to predict what events it will generate, and schedule these events early. Lookahead may, in general, be difficult to incorporate into certain classes of simulations, but nevertheless is very important for simulations requiring guaranteed message ordering services to achieve acceptable performance. As demonstrate in (Fujimoto 1989), lookahead can have a very dramatic effect on federation performance.

Lookahead is clearly very intimately related to details of the simulation model, and thus cannot be determined automatically by the RTI. Some examples of where lookahead may be derived are described below.

- *Physical limitations concerning how quickly one federate can react to an external event.* Suppose the minimum amount of time for a tank to respond to an operator's command (e.g., to fire an ordnance) is 500 milliseconds. This means the simulation can guarantee that it will not schedule the results of any new operator actions until at least 500 milliseconds into the future, providing a lookahead of this amount.
- *Physical limitations concerning how quickly one simulation can affect a second simulation.* Suppose two tanks are ten miles apart, and there is also a maximum speed of a projectile fired from one tank to another. These constraints place a lower bound on how much time must elapse for the first tank to affect the second. Thus, events such as a projectile exploding at the second tank can be scheduled into the future, providing some degree of lookahead.
- *Tolerance to temporal inaccuracies.* Suppose a simulation produces an event at time T, but assigning a time stamp of T+1 second does not impact the accuracy of the simulation results. Then, the simulation may schedule events 1 second into the future, providing a lookahead of this amount.
- *Time stepped simulations:* In a time stepped simulation, the lookahead is normally the size of the time step. This is because a simulation can only schedule events into the next time step (or later), but not into the current time step.
- *Non-preemptive behavior.* Suppose a tank is moving north at 30 miles per hour, and nothing in the federation model could cause it to change any events produced by the tank over the next ten minutes. These events could therefore be scheduled immediately, resulting in a lookahead of 10 minutes.

Lookahead can change dynamically during the simulation. However, lookahead cannot instantaneously be reduced. At any instant, a lookahead of L indicates to the RTI that the federate will not generate any event (using timestamp ordering) with timestamp less than C+L, where C is the federate's current logical time. If the lookahead is reduced by K units of time, the federate must advance K units before this changed lookahead can take effect, so no events with timestamp less than C+L are produced. In the HLA, a single lookahead value is declared by each federate. This value may change at runtime, but reductions in lookahead do not take effect immediately.

6.4 Zero Lookahead and Simultaneous Events

When a federate receives a **Time Advance Grant** to simulation time T as the result of invoking the **Next Event Request** or **Time Advance Request** service, the RTI guarantees the federate has received all events with time stamp equal to T. This is a useful guarantee because it means the federate can now order the events at time stamp T and process them without fear of later receiving additional events containing this time stamp. This guarantee cannot be kept if federates have a lookahead value of zero. This is because the federate that has received a **Time Advance Grant** to logical time T could now generate a new event with time stamp equal to T. This, in turn, might result in a

second event also with time stamp equal to T that is sent back to the original federate. Early versions of the time management services circumvented this problem by requiring all federates to have strictly positive lookahead values.

During the HLA prototyping phase, it became apparent that prohibiting zero lookahead values was too restrictive. To support zero lookahead federates, but without giving up the property that federates could guarantee that they had received all events at a given time stamp (so that the events could be ordered in a deterministic fashion), two new services were added. These services are called **Next Event Request Available (NERA)** and **Time Advance Request Available (TARA)**. NERA (TARA) is similar to NER (TAR) except in one important aspect: when a **Time Advance Grant** to logical time T is issued by the RTI in response to a NERA (TARA) service call, the RTI does *not* guarantee that the federate has received all messages with time stamp exactly equal to T. The federate is able to generate new messages with time stamp equal to T (i.e., zero lookahead messages). By contrast, when a federate receives a **Time Advance Grant** to time T as the result of a NER or TAR call, the federate is prohibited from sending messages with time stamp exactly equal to (or less than, of course) T, even if the federate's lookahead is zero. This approach allows federates to generate zero lookahead messages, but also gives the federate the ability to determine when it has received all messages with time stamp equal to the current logical time.

6.5 Guarantees on the Time Stamp of Outgoing Messages

Consider again the snap shot shown in Figure 4, but assume each federate has a lookahead value of zero. Suppose each federate is event driven and each has a single local event (not shown in Figure 4) with time stamp 20. Federates A, B, and C will be blocked on a call to NER with time parameter 20. Suppose federate D completes processing the time stamp 10 event, and also calls NER(20). If the only constraint on the time stamp of new messages is the time stamp must be greater than or equal to the federate's current time plus lookahead, the RTI cannot preclude the generation of new messages with time stamp value 20. Thus, the RTI cannot issue a **Time Advance Grant** to any federate, and the simulation is deadlocked!

The solution to this problem can be found by considering the operation of a *sequential* event-oriented simulation program. The simulator can always advance to the time stamp of the next event in its list of pending events after processing each event. This is because the simulator can guarantee that at the instant it has completed processing an event, no new events will be generated with time stamp less than the time stamp of the next event in the event list.

Unfortunately, this same guarantee cannot be made in a distributed simulation. Consider again federate D in Figure 4. The smallest time stamp of any event in the snapshot destined for federate D is 13. However, there is nothing to prevent federate C from generating a new event for federate D with time stamp 11 during the course of processing its next event. Thus federate D cannot advance to logical time 13. However, because each federate uses an event driven time flow mechanism, we can say that the smallest time stamp on any unprocessed event in the entire federation is 11, so it can be

guaranteed that the smallest time stamp of any future event is also 11. This fact would enable an event, and **Time Advance Grant** (to logical time 11) to be issued to federate C, thus breaking the deadlock.

This logic is included in the following guarantees made by federates to the RTI concerning the time stamp of any events the federate might generate in the future. Below, a *pending* NER/NERA/TAR/TARA refers to an invocation of this service where no **Time Advance Grant** has been issued.

- A federate cannot generate any new messages with time stamp less than the federate's current logical time plus its lookahead. Further, if the federate reached the current logical time via the NER or TAR service, the federate is also prohibited from generating a new message with time stamp equal to its current time, even if its lookahead is zero.
- If a federate has a pending NER with time parameter T, it guarantees any new messages it generates will have time stamp greater than or equal to T plus the federate's lookahead (or strictly greater than T if the federate's lookahead is zero), *provided* no additional TSO messages with time stamp less than T are delivered to the federate. If the RTI does deliver a new message to the federate with time stamp less than T, this guarantee is declared null and void. If the federate has a pending NERA call the same guarantee is made except the federate may still generate messages with time stamp equal to T if its lookahead is zero.
- If the federate has a pending TAR service call with time parameter T, the federate unconditionally guarantees that any new messages it generates will have a time stamp greater than or equal to T plus the federate's lookahead (strictly greater than if the federate's lookahead is zero), regardless of what new messages are delivered by the RTI in the future. A similar guarantee is made if the federate has a pending TARA call, except the federate can generate new messages with time stamp T if its lookahead is zero.

These rules form a contract between the federate and the RTI. This contract is important because it enables the RTI to determine the minimum time stamp of any new message the federate can later produce, which is essential in computing LBTS values.

6.6 Optimistic Event Processing

The discussion thus far has focused entirely on so-called *conservative* synchronization techniques that avoid the possibility of a federate processing events out of time stamp order. The other well-known approach to synchronization is the so-called *optimistic* technique that allow messages to be processed out of time stamp order, but use some mechanism to recover. Jefferson's Time Warp mechanism is the most well known optimistic synchronization protocol (Jefferson 1985).

The HLA supports optimistic federates while still maintaining time management transparency. Specifically, the HLA time management services do not require all federates to support a rollback and recovery capability even if one federate is using optimistic event processing. Indeed, it is envisioned that federations may include *both*

optimistic and conservative federates within a single execution. Conservative federates not needing or desiring to utilize optimistic processing techniques may completely ignore the optimistic time management services with no ill effects.

An important goal of the optimistic time management services is to enable optimistic execution among a collection of optimistic federates. Thus, simple solutions such as requiring that the optimistic federate *only* send messages that it can guarantee will not be later canceled are undesirable, because they do not fully exploit the potential offered by optimistic execution.

A federate executing on a parallel processor using the Time Warp protocol will allow some simulation events to be processed, even though it is possible another event containing a smaller time stamp will later arrive, possibly from a different federate. If such an event later arrives, some event computations must be rolled back. Realizing rollback means rolling back the state of the simulation process receiving the late event and canceling any new events scheduled by the rolled back computation. If the canceled event has already been processed at the destination process, that process is also rolled back, possibly generating additional event cancellations. Recursively applying this procedure will eventually erase the effects of the incorrect computation.

To support this style of execution, the following capabilities are required:
- Federates must be able to receive TSO messages *before* the RTI can guarantee that no smaller time stamped messages will be later received, i.e., before the RTI can guarantee message ordering, in order to allow optimistic event processing. In the HLA this ability is provided by the **Flush Queue Request** service that forces the RTI to deliver all buffered events in the destination RTI's internal queues.
- Federates must be able to cancel previously sent messages. This is accomplished via the HLA's **Retract** primitive. If the cancelled message has already been delivered to the destination federate, the cancellation request is forwarded to that federate. At this point, it is the destination federate's responsibility to roll back, and possibly generate additional retraction requests.
- Optimistic federates must be able to compute a lower bound on the logical time of any future roll back. This lower bound, called Global Virtual Time (GVT) allows the optimistic federate to reclaim memory resources, and to perform operations such as I/O that cannot be rolled back. In Time Warp, rollbacks are caused by receiving a message or event cancellation in the past of the receiving federate. Thus the federate's LBTS value provides the information necessary for the federate to compute GVT.

To illustrate the use of these services in an optimistic federate, the following outlines how a Time Warp based federate (TW) could be included in an HLA federation:
- The TW federate uses the **Flush Queue Request** service to receive, and optimistically process events.
- Optimistically generated messages are transmitted through the RTI to other federates, the same as ordinary, non-optimistic messages. The RTI does not distinguish between these two types of messages.

- The event retraction primitive provided by the RTI is used to cancel optimistic messages that later prove to be incorrect.
- If the message for the canceled event has not been delivered by the RTI to the receiving federate, the message is cancelled within the RTI. If the message has already been delivered to the receiving federate, the retraction request is forwarded to the federate which must perform the cancellation itself, typically by performing a rollback in the receiving federate, possibly generating additional cancellation (retraction) requests.
- From the perspective of the Time Warp federate, global virtual time (GVT) is the logical clock of the Time Warp federate. This, plus the Time Warp federate's lookahead, gives a lower bound on the time stamp of messages that may be generated in the future by that federate. This information enables the RTI's LBTS computation to prevent conservative federates from receiving optimistic messages until it has been guaranteed that they will not be later cancelled. Any event with time stamp less than GVT is guaranteed not to be prone to future rollbacks, and since events must be generated at least L time units into the future, where L is the lookahead, any message with time stamp less than GVT+L is guaranteed not to be subject to any future cancellation.

The "main loop" of a typical Time Warp simulator might execute the following steps:

```
while (simulation execution is still in progress) {
        Next_Event_Time = time stamp of next local event
        invoke Flush Queue Request (Next_Event_Time)
        honor RTI requests for Reflect Attribute Values, Receive Interaction
                or event retractions (cancellations). Place these incoming messages into
                queues within the federate, process any rollbacks or annihilations, using
                message retraction to cancel previously sent messages
        honor RTI service request for Flush Queue Grant (Local_LBTS)
        fossil collect using Local_LBTS as the GVT value
        process next smallest time stamped message(s)
        }
```

The last statement will typically process messages containing a time stamp larger than GVT, giving rise to optimistic execution. Any number of messages may be processed before the RTI queues are again flushed. Also, it is perhaps worth pointing out that fossil collection need not be performed every iteration through the main processing loop.

One attractive property of this approach is it enables Time Warp federates to "plug into" the RTI, without any other federate (even optimistic ones) realizing there is an optimistic federate in the federation execution. The RTI automatically allows for optimistic exchange of messages among a collection of optimistic federates, and at the same time, guarantees that optimistic messages are not released to conservative federates, all transparent to the federates in the federation. Further, no special GVT messages must be exchanged between optimistic federates, as the RTI automatically provides the

information necessary for each optimistic federate to compute GVT locally. Finally, another attractive feature of this approach is it requires only modest modification of the "conservative" time management services already specified in the RTI.

A well known problem in optimistic simulations is "throttling" the federate to prevent it from executing too far ahead into the future, leading to inefficient execution. At present, the HLA does not specify what throttling mechanisms should be used; this is currently he responsibility of individual federates.

7. Conclusions

The time management services of the HLA are central to achieving interoperability among analytic simulations using event driven or time stepped mechanisms to advance simulation time. The time management system includes mechanisms to ensure time stamp ordered delivery of messages, we well as mechanisms for federates to advance logical time so that the federate does not receive messages with time stamp less than its current logical time. Time management transparency so that one federate need not understand the local time management structure of other federates is key to achieve interoperability among federates using different local time management mechanisms.

8. Additional Readings

The HLA time management design is based on a handful of key principles developed largely by the parallel discrete event simulation community over the last twenty years. Specifically, computation of LBTS values, lookahead, and potential deadlock problems were recognized in the earliest time management algorithms (Bryant 1977; Chandy and Misra 1978; Peacock, Wong et al. 1979). The importance of lookahead and its impact on performance is discussed in (Fujimoto 1989). The need to pass information concerning the time stamp of the next local event was perhaps first used in (Chandy and Misra 1981), and exploited heavily in second generation time management algorithms, e.g., (Lubachevsky 1989). Optimistic time management for distributed simulations were first introduced in (Jefferson 1985). An in depth treatment of time management techniques is presented in (Fujimoto 1999).

9. ACKNOWLEDGMENTS

10. References

Bryant, R. E. (1977). Simulation of Packet Communication Architecture Computer Systems. Computer Science Laboratory. Cambridge, Massachusetts, Massachusetts Institute of Technology.

Chandy, K. M. and J. Misra (1978). "Distributed Simulation: A Case Study in Design and Verification of Distributed Programs." IEEE Transactions on Software Engineering **SE-5**(5): 440-452.

Chandy, K. M. and J. Misra (1981). "Asynchronous Distributed Simulation via a Sequence of Parallel Computations." Communications of the ACM **24**(4): 198-205.

DIS Steering Committee (1994). The DIS Vision, A Map to the Future of Distributed Simulation. Orlando, Florida, Institute for Simulation and Training.

Fujimoto, R. M. (1989). "Performance Measurements of Distributed Simulation Strategies." Transactions of the Society for Computer Simulation **6**(2): 89-132.

Fujimoto, R. M. (1999). Parallel and Distributed Simulation Systems, Wiley Interscience.

Jefferson, D. (1985). "Virtual Time." ACM Transactions on Programming Languages and Systems **7**(3): 404-425.

Lubachevsky, B. D. (1989). "Efficient Distributed Event-Driven Simulations of Multiple-Loop Networks." Communications of the ACM **32**(1): 111-123.

Mehl, H. (1992). A Deterministic Tie-Breaking Scheme for Sequential and Distributed Simulation. Proceedings of the Workshop on Parallel and Distributed Simulation. M. Abrams and P. Reynolds, Jr., Society for Computer Simulation. **24:** 199-200.

Peacock, J. K., J. W. Wong, et al. (1979). "Distributed Simulation Using a Network of Processors." Computer Networks **3**(1): 44-56.

Wilson, A. L. and R. M. Weatherly (1994). The Aggregate Level Simulation Protocol: An Evolving System. Proceedings of the 1994 Winter Simulation Conference**:** 781-787.