HLA Time Management: Design Document
Version 1.0
August 15, 1996

This document describes the recommendations of the time management working group concerning functionality and design of the time management component of the HLA. Specifically, the recommended services provided by the RTI related to time management are described, their envisioned use, and in some cases, possible approaches for implementation. This is an evolving, working document that describes both the current status of the design and directions for future evolution.

1. Overview

A federation execution can be viewed as a collection of federates, each performing a sequence of computations. Some of these computations are referred to as *events*, and some of these events are relevant to other federates. The RTI notifies other federates that have indicated an interest in an event by sending *messages* to them containing information concerning the event.

Consider a real-time (e.g., a human-in-the-loop or a hardware-in-the-loop) federation execution, a principal application domain for the HLA. In an ideal world, delays associated with the execution of modeling computations and latencies in the communication network during the federation execution would be identical to delays in the system being modeled, resulting in a precise emulation of that system. Actually, federation latencies could be smaller than those in the real world since one can always introduce additional delay. Unfortunately, it is often the case that latencies in the federation execution are larger than those in the real world. Worse, federation latencies are *not* uniformly larger. This can lead to temporal distortions (or anomalies) causing the simulated world to deviate from the real world in undesirable ways. For instance, cause-and-effect relationships in the real world may be distorted, resulting in a federation execution where the "effect" appears to happen before the "cause."

The goal of the time management architecture is to define an infrastructure to allow federations to reduce the occurrence and effect of these anomalies where this is necessary to meet the objectives of the federation execution. In most implementations, this added functionality will usually come at increased cost, e.g., higher latency, higher network bandwidth consumption, and/or additional computational resources that might have been used for other purposes.

There are essentially two related problems that need to be solved by the time management system:

1. things don't happen during the federation execution when they are supposed to happen, and
2. things don't happen in the order that they are supposed to happen.

The first problem cannot, in general, be solved, simply because the computers/networks used in the execution may not be fast enough. The second problem can be solved, provided the order in which the events are supposed to occur can be precisely defined by the federates. The approach used in the HLA time management architecture is to enable complete solution of the second problem and at the same time, enable solution to the first problem to the extent that this is possible. Further, a principal objective of the time management structure is to allow event

ordering to be relaxed, or ignored altogether when this functionality is not required or the associated performance penalties cannot be tolerated.

The HLA time management architecture attacks the ordering problem by defining a *logical time clock*. Logical time is roughly synonymous with "simulation time" in the classical discrete event simulation literature, and is used to ensure that federates observe events in the same order (though not necessarily at the same instant) that the corresponding activities would be observed in the physical world. The HLA time management architecture also provides a means for federates to coordinate their activities using the *wallclock* but does not provide mechanisms for accomplishing this coordination.

2. Interoperability

The High Level Architecture time management (HLA-TM) structure is intended to support interoperability among federates utilizing different internal time management mechanisms. For example, HLA-TM should support within a single federation execution:

1. interoperability among federates with different message ordering requirements, e.g., a single federation execution might include DIS federates that need not process event notices in time stamp order, and ALSP federates that do.
2. interoperability among federates using different internal time advance (time flow) mechanisms, e.g., time-stepped federates, event driven federates, and federates using "independent" time advance (e.g., DIS) mechanisms.
3. interoperability among scaled real-time (constrained) and as-fast-as-possible federates, assuming individual federates executing in conjunction with the RTI perform at least as fast as scaled wallclock time.
4. interoperability among federates using conservative and optimistic synchronization protocols, e.g., Time Warp [Jeff85] simulations, executing on multiprocessors, might be combined with other parallel simulations using conservative synchronization (e.g., see [Chan79]).
5. individual federates using a mixture of message ordering and transportation services, e.g., future DIS systems using ordered, reliable message delivery for certain types of events (e.g., weapon detonations) intermixed with receive ordered, best-effort delivery for other events where order and guaranteed delivery are less critical (e.g., position updates). Allowing different events within a single federate to utilize different categories of message delivery service enables gradual, evolutionary exploitation of previously unused HLA-TM services.

The HLA supports these capabilities provided federates adhere to certain requirements necessary to realize each service, e.g., lookahead (discussed later) is required to provide time stamp ordered delivery of messages. Further, individual federates executing in conjunction with the RTI must deliver real-time performance in federations whose execution is paced by wallclock time. In particular, lookahead plays an important role in determining the performance of federations requiring time stamp ordered message delivery.

To achieve these goals, a *single*, unifying approach to time management is developed to provide time management interoperability among disparate federates. Different categories of simulations are viewed as special cases in this unified structure, and typically use only a subset of the RTI's full capability. Federates need not explicitly indicate to the RTI the particular time flow

mechanism (time stepped, event driven, independent time advance) being used within the federate.

3. Definitions

A few key definitions that apply throughout this document are enumerated below. A glossary containing a more complete set of definitions is included at the end of this document. Among the definitions concerning time, wallclock (and scaled wallclock), logical, and federate time all specify points on a global federation time axis, and are important to understand the remainder of this document.

current time (of a federate): same as federate time.

event: A change of object attribute value, an interaction between objects, an instantiation of a new object, or a deletion of an existing object that is associated with a particular point on the federation time axis. Each event contains a time stamp indicating when it is said to occur.

federate time: time that the federate has coordinated with the RTI (if the federate uses logical time) or an internally calculated time like scaled wallclock time (if the federate does not use logical time). Federate time is synonymous with the "current time" of the federate. At any instant of an execution different federates will, in general, have different federate times.

federation time axis: a totally ordered sequence of values where each value represents an instant of time in the physical system being modeled, and for any two points T_1 and T_2 on the federation time axis, if $T_1 < T_2$, then T_1 represents an instant of physical time that occurs before the instant represented by T_2. The progression of a federate along the federation time axis during an execution may or may not have a direct relationship to the progression of wallclock time.

logical time axis: a set of points (instants) on the federation time axis used to specify before and after relationships among events.

logical time: A federate's current point on the logical time axis. If the federate's logical time is T, all time stamp ordered (TSO) messages with time stamp less than T have been delivered to the federate, and no TSO messages with time stamp greater than T have been delivered; some, though not necessarily all, TSO messages with time stamp equal to T may also have been delivered. Logical time does not, in general, bear a direct relationship to wallclock time, and advances in logical time are controlled entirely by the federates and the RTI. Specifically, the federate requests advances in logical time via the **Time Advance Request** and **Next Event Request** RTI services, and the RTI notifies the federate when it has advanced logical time explicitly through the **Time Advance Grant** service. Logical time may be used to determine the current time of the federate (see definition of federate time). Logical time is only relevant to federates using time stamp ordered message delivery and coordinated time advances, and may be ignored by other federates.

scheduling an event: invocation of a primitive (**Update Attribute Values, Send Interaction, Instantiate Object**, or **Delete Object**) by a federate to notify the RTI of the occurrence of an event. Scheduling an event normally results in the RTI sending messages to other federates to notify them of the occurrence of the event.

time management: a collection of mechanisms and services to control the advancement of time within each federate during an execution in a way that is consistent with federation requirements for message ordering and delivery.

time stamp (of an event): a value representing a point on the federation time axis that is assigned to an event to indicate when that event is said to occur. Certain message ordering services are based on this time stamp value. In constrained simulations (executions paced by wallclock time), the time stamp may be viewed as a deadline indicating the latest time at which the message notifying the federate of the event may be processed.

wallclock time: a federate's measurement of true global time (defined in the glossary), where the measurand is typically output from a hardware clock. The error in this measurement can be expressed as an algebraic residual between the wallclock time and true global time or as an amount of estimation uncertainty associated with the wallclock time measurement software and the hardware clock errors.

4. Assumptions

The following assumptions apply throughout this document:

1. No common, global, clock is assumed. At any instant in the execution, different federates will usually have different current time values (see the definition of federate time). Even for (scaled) real-time federates that do not utilize logical time, drift between different clocks may result in different wallclock times for different federates at any instant in the execution, unless the federates are engineered such that the maximum clock drift is an order of magnitude smaller than the finest time representation granularity in use by the federation. Thus, statements such as "the federation is now at time X" are usually meaningless because at any instant, there is usually no single point on the federation time axis recognized by all federates as their current time. Rather, a federate whose current time is X *perceives* the federation to be at time X, but at any instant, different federates will in general perceive the federation to be at different times.

2. Each change in the state of an object (referred to as an event, this includes attribute updates, interactions, instantiates, and delete actions) is assigned a time stamp that is determined by the federate "scheduling" (notifying the RTI of) the change. A consequence of this assumption is that federates may generate events with time stamps "in the future", i.e., time stamps larger than the current time.

3. Federates utilizing logical time may not schedule events with a time stamp "in the past," i.e., with time stamp smaller than the current time of the federate (i.e., its federate time).

4. Federates need *not* generate events in time stamp order. For example, a federate may first schedule an event with time stamp 10, then later schedule a new event (possibly resulting in a message being sent to the same destination as the first) with time stamp 8.

5. HLA Time Management Services

Time management is concerned with the mechanisms for controlling the advancement of time during the execution of a federation. Time advancement mechanisms must be coordinated with other mechanisms responsible for delivering information (e.g., notices of attribute updates and interactions) to individual federates because such information is time stamped to indicate when the information is valid. For example, federates may require that no information be received "in the federate's past," i.e., with time stamp less than the current time of the federate. Thus, the

time management services supported in the HLA must encompass two aspects of federation execution:

- *Transportation services:* Different categories of service are specified that provide different reliability, message ordering, and cost (latency and network bandwidth consumption) characteristics.
- *Time advancement services:* Different primitives are provided for federates to request advances in logical time. A simple protocol is provided to enable a federate to control the flow of attribute updates and interaction requests to that federate.

The different categories of transportation service are distinguished according to (1) reliability of message delivery, and (2) message ordering. With respect to reliability, *reliable message delivery* means the RTI utilizes mechanisms (e.g., retransmission) to increase the probability that the message is eventually delivered to the destination federate. This improved reliability normally comes at the cost of increased latency. On the other hand, the *best effort message delivery* service attempts to minimize latency, but with the cost of lower probability of delivery. Other types of reliability are possible, but are only indirectly related to time management. Of greater importance to time management are message ordering characteristics that specify the order and time at which messages may be delivered to federates. These are described in detail next.

Time advance primitives provide the means for federates to coordinate their logical time advances with the time stamp of incoming information, if this is necessary. The time advance mechanism in the RTI must accommodate both real-time, scaled real-time, and as-fast-as-possible executions. The HLA time advance mechanisms are described later.

6. Message Ordering

Central to the HLA time management services are mechanisms to order messages that are passed to federates. A variety of services are provided to support interoperability among federates with diverse requirements. Five ordering mechanisms are currently specified in the HLA: receive, priority, causal, causal and totally ordered, and time stamp order. These provide, in turn, increased functionality but at increased cost.

Each federate may intermix different message ordering services for different types of information within a single federation execution. For example, position updates where reliable delivery and ordering are not important may utilize a best effort, receive order category of service. These messages may be intermixed with messages for ordnance detonation events utilizing reliable, time stamp ordered delivery.

6.1. Receive Order

This is the most straightforward, lowest latency ordering mechanism. Messages are passed to the federate in the order that they were received. Logically, incoming messages are placed at the end of a first-in-first-out (FIFO) queue, and are passed to the federate by removing them from the front of this queue.

Receive order should be utilized by applications where minimizing communication latency is more important than adhering to message orderings that guarantee causality. Hard real-time users will normally exploit this ordering and insert time in the “user specified tag” argument

when their federates invoke the **Send Interaction** and **Update Attribute Values** RTI services or when their federates respond to **Receive Interaction** and **Reflect Attribute Values** calls from the RTI. Federates currently using DIS protocols will typically also utilize this order, though it may be beneficial to use other message orderings for certain types of events.

6.2. Priority Order

Incoming messages are placed in a priority queue, with the message time stamp used to specify its priority. Messages are passed to the federate lowest time stamp first. In other words, the RTI attempts to deliver messages in time stamp order based on the local information available to the RTI when the message is delivered. However, a message could later arrive and be delivered to the federate that has a time stamp smaller than one that has already been delivered. Further, this service does not prevent a message from being delivered to a federate in its "past" (time stamp less than the federate's current time). While this service does not guarantee time stamp ordered delivery, it is less costly in terms of latency and synchronization overhead than the service guaranteeing time stamp ordered delivery. This ordering should *not* be used if time stamp ordering of messages is essential to the correct operation of the federate.

Priority order with best effort delivery may be used for federates where sequences of messages require ordering, but the increased latency associated with either reliable delivery or guaranteed order cannot be tolerated. For example, speech packets may utilize this service.

Messages using receive or priority order are available for delivery to the federate as soon as the message has been received. By contrast, the RTI may buffer messages using causal or time stamp order until it can guarantee the desired ordering properties.

6.3. Time Stamp Order

Messages utilizing this service will be delivered to federates in time stamp order. The **Instantiate Discovered Object**, **Remove Object**, **Receive Interaction** and **Reflect Attribute Values** calls by the RTI to the federate will not occur (thus preventing delivery of messages using this ordering) until the RTI can guarantee that no message utilizing time stamp ordered (TSO) delivery with a smaller time stamp will later be received. To accomplish this task, the RTI will hold incoming messages in its internal queues until it can preclude subsequent arrival of another TSO message containing a smaller time stamp. Federates are *not* constrained to generate messages in time stamp order. A conservative parallel simulation style synchronization protocol is used to implement this service.

In addition to delivering messages in time stamp order, the RTI also ensures that no message is delivered to a federate "in its past," i.e., no message is delivered that contains a time stamp less than the federate's current logical time. This is accomplished by forcing the federate to explicitly request advances in logical time using the RTI's time advance services. The RTI will not provide a "grant" to the time advance request until it can guarantee that no messages containing a time stamp smaller than the time of the grant will later be received.

This time stamp order service is required for classical discrete event simulations (e.g., constructive simulations) where time stamp order event processing is the norm. Federates that do not traditionally utilize time stamp ordered event processing, e.g., DIS simulations, but could be enhanced with strong event ordering properties for certain types of information may also

utilize this service, while continuing to use less costly message ordering services for other information where order is less critical.

An important feature of the time stamp order service is that all federates receiving messages for a common set of events will receive those messages in the same order, i.e., a total ordering of events is provided. This eliminates certain temporal anomalies that might otherwise occur when different federates perceive different orderings of events. The RTI provides a consistent tie breaking mechanism so messages containing identical time stamps will be delivered to different federates in the same order. Further, the tie-breaking mechanism is deterministic, meaning repeated executions of the federation will yield the same relative ordering of these events if the same initial conditions and inputs are used, and all messages are transmitted using time stamp ordering.

6.4. Causal Order

Like the time stamp order service, the causal order service ensures that messages are delivered to federates in an order that is consistent with before and after relationships of the events represented by these messages. For example, firing a weapon must occur before the target is destroyed. Unlike the receive and priority ordered services where varying communication latencies could cause the message for the destroy event to be delivered before the message for the fire event, the causal and time stamped ordered services guarantee that if one event "happens before" a second, then all federates will receive messages for the first prior to the messages for the second.

The fundamental difference between the causal and time stamp ordered services is concerned with their respective definitions of the "happens before" relationship. In the time stamped order service, an event is said to happen before another event if it has a smaller time stamp than the second, following one's intuitive notion of event ordering in physical systems. The federation can precisely specify which events happen before which other events by assigning appropriate time stamps, e.g., the "fire" event might be assigned a time stamp of 102, and the "destroy" event a time stamp of 103 to specify that "the fire event happens before the destroy event."

In the causal ordered service, the "happens before" relationship is that defined by Lamport [Lamp78]. The execution of each federate can be viewed as an ordered sequence of "actions" (e.g., execution of a single machine instruction can be viewed as an action). Two specific actions of particular interest are sending and receiving a message. Lamport defines the happens before relationship as follows: (i) if actions A and B occur in the same federate, and A appears before B in the ordered sequence of actions within that federate, then A happens before B (ii) if A is the action sending a message to another federate, and B is the action receiving the same message in the second federate, then A happens before B, (iii) if A happens before B, and B happens before C, then A happens before C (transitivity). The happens before relationship can be easily extended to include messages: (i) if a message X is sent by a federate before the same federate sends another message Y, then X happens before Y, (ii) if X is received by a federate before that federate sends another message Y, then X happens before Y, (iii) if X happens before Y, and Y happens before a third message Z, then X happens before Z. An event in the HLA is said to causally precede another event if the first event happens before the second, using the "happens before" relationship defined above.

The HLA causal event ordered service guarantees that if an event E causally precedes another event F and messages for both events are delivered to a federate, then the message for E will be delivered to that federate before the message for F.

For example, consider the example depicted in Figure 1 where federate A fires a missile at federate B, destroying an entity in that federate, and a third federate C observes this exchange. The fire event in federate A generates messages to federates B and C. Upon receiving this message, federate B generates a state update event indicating that an entity it contains has been destroyed. In the scenario depicted in this figure, the "destroy" message reaches C before the "fire" message. Because the "fire" event causally precedes the "destroy" event, the causal event ordering service guarantees that the message for the fire event is delivered to C prior to the message for the destroyed event. The RTI will delay delivery of the message for the "destroy" event until after it receives and delivers the message for the "fire" event to C. The receive and priority ordered services do not guarantee causal ordering, so they could deliver the message for the destroy event before the message for the fire event in this scenario.

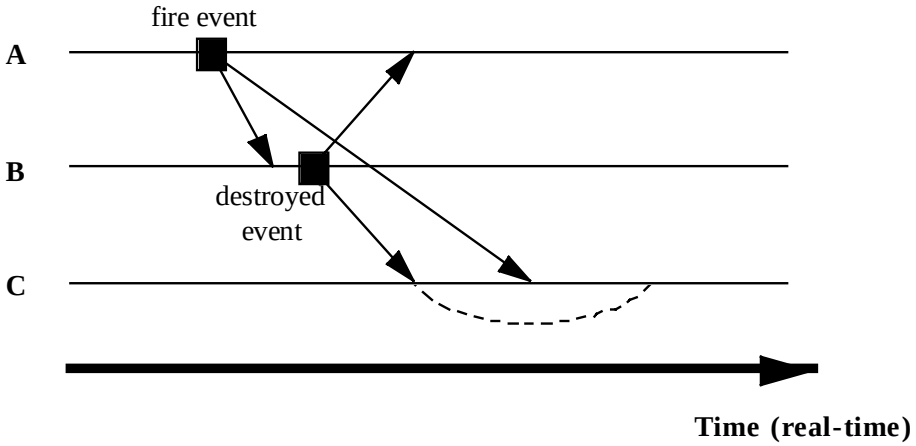

Figure 1. Scenario demonstrating causal event ordering. The RTI delays delivery of the first message received by C.

In the basic causal ordered service, messages corresponding to events that are *not* causally related (referred to as concurrent events) may be delivered to federates in any order. A variation on causal ordering is to guarantee that all federates receive messages for concurrent events *in the same order*, thereby defining a total ordering of events. This service is commonly referred to as CATOCS (causally and totally ordered communications support) in the literature. Algorithms for implementing CATOCS have been developed and implemented (e.g., see [Birm91]).

Figure 2 shows an example where causal and total ordering may be beneficial. Two federates (A and B) modeling enemy aircraft are taking off from an air field. Two pilots are assigned to intercept, with pilot 1 (federate C) given orders to attack the first enemy aircraft to take off, and pilot 2 (federate D) assigned to attack the second. Assume the "take-off" events are concurrent, e.g., the aircraft are taking off from different runways. Without total ordering, messages for the take-off events may arrive at the two federates in different orders. Figure 2 shows a scenario where pilot 1 incorrectly believes aircraft 2 took off first, while pilot 2 correctly believes aircraft 1 took off first. The end result is both pilots attack aircraft 2! CATOCS would circumvent this anomaly by ensuring that both pilots see the same ordering of events. Both pilots may perceive an incorrect order (e.g., both might believe aircraft 2 took off first if the messages from Federate

A are delayed), but the result of this error is likely to be less severe than the previous scenario where both pilots attack the same aircraft.

6.5. Contrasting Time Stamp and Causal Order

It is important to understand the differences between time stamp and causal order in order to determine which is appropriate for specific types of information. Time stamp order provides more stringent ordering services than causal ordering. Both the time stamp and causal ordering guarantee that messages for causally related events are delivered to federates in the order dictated by the causal happens before relationship. However, the ordering of concurrent events in the causal ordered service, even if CATOCS is used, is non-deterministic because it depends on latencies within the communication network. Thus, causal ordering, with or without total ordering, is *not* sufficient to produce repeatable results without the use of previously generated logs to specify message ordering. Time stamp ordering must be used if this is a requirement.

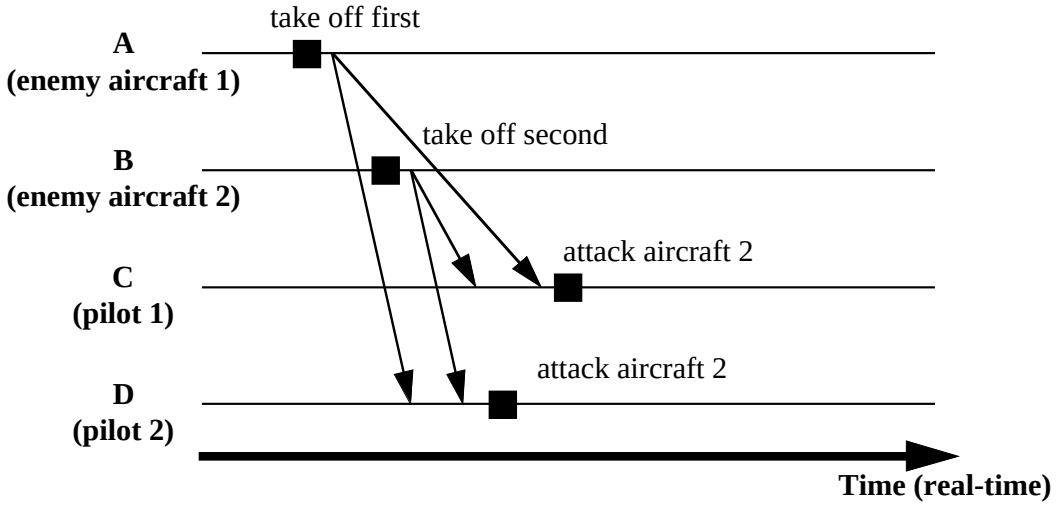

Figure 2. Scenario demonstrating causal and total ordering. Without total ordering, federates C and D may see the two take-off events in different orders, resulting in an anomaly.

Further, while causal ordering is sufficient to avoid certain anomalies (e.g., receiving a message for a tank destroyed event before the message for the event indicating the tank has been fired upon), it is not sufficient in other situations, at least not without the introduction of additional messages to enforce ordering constraints. Specifically, causal order is not sufficient if ordering relationships among *concurrent* events are important, or if there are "hidden dependencies" between events, as elaborated upon below.

Ordering Concurrent Events. Consider an execution that includes three federates, each representing a tank. Suppose tank A has orders to fire upon the first target to come within range of its cannon. It may happen that tank B comes within range before tank C. However, because state updates by B and C are concurrent events, causal ordering does not guarantee that B's state update message will reach A before C's update message. This could cause A to incorrectly fire upon C. If capturing this behavior is important to the objectives of the federation, the time stamp order service should be used rather than the causal order service. Time stamp order will produce correct results because B's update indicating it enters A's range will have a smaller time stamp than C's update, so B's message will be delivered first.

Hidden Dependencies. Consider a battle in a military campaign that is staged as a timed sequence of actions, e.g., a diversion might be initiated by one unit at time 100, followed by initiation of the actual attack by another unit at time 150. It is clear that the commanders planning the operation staged its execution so that the diversion occurs before the main attack. However, CATOCS does not guarantee that messages corresponding to the diversion reach federates representing the opposing force before messages corresponding to the main attack! The problem is that all message ordering in CATOCS is based *only* on messages passed between federates, so semantic relationships between events are not visible to the RTI. Again, this is an instance where time stamp ordering should be used to ensure federates receive messages for events in the correct time sequence.

In principal, both of the above problems could be addressed using causal ordering by adding additional messages to explicitly specify dependencies between events. However, in general, determination of what messages to send and when to send them may not be straight-forward.

The principal advantage of causal ordering relative to time stamp order is that it does not require specification of lookahead (discussed later). Thus, causal order may provide an acceptable alternative to time stamp ordering for federates with little lookahead where use of optimistic (rollback-based) event processing techniques is not considered viable. Further, it is anticipated that in most implementations, causal message order will typically yield lower communication latency and require less bandwidth to implement than time stamp order.

6.6. Lookahead

The time stamp order service requires specification of a quantity called lookahead. To motivate the need for lookahead, consider an as-fast-as-possible execution where time stamp order is specified for all communications. Consider the federate with the *smallest* logical time at some instant in the execution. Let this federate have a logical time of T. This federate could generate events relevant to every other participant in the federation with a time stamp of T. This implies the RTI cannot deliver any message with time stamp larger than T to any federate. In turn, this implies no federate can advance its logical clock beyond T because it is then prone to receiving notification of an event in its past.

This is a well-known problem that has been widely studied in the parallel simulation community. The two principal approaches to circumventing it are:

- use the notion of lookahead, described below, to define conservative protocols that prevent out-of-order delivery of messages, or
- use optimistic synchronization techniques that allow messages to be delivered out of time stamp order, and use a rollback mechanism within the federate to recover from errors introduced from out-of-order delivery.

HLA-TM supports both of these approaches. Optimistic synchronization is described later. Lookahead is elaborated upon next.

If one were to mandate that *no federate may schedule an event with time stamp less than the federate's current time plus a value L,* then the RTI can allow concurrent delivery and processing of messages in a time window L time units wide beginning at the minimum logical

time of any federate. This value L is referred to as the lookahead for the federate because it must be able to "look ahead" L time units into the future, or in other words, predict attribute updates and interactions at least L time units "ahead of time". Lookahead may, in general, be difficult to incorporate into certain classes of simulations, but nevertheless is very important for federates requiring guaranteed message ordering services to achieve acceptable 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 federate 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 federate can affect a second.* 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 federate produces an event at time T, but the receiver of the message for that event cannot distinguish between the event occurring at time T and T+100 milliseconds (e.g., in a training simulation, it might be the case that a muzzle flash occurring at time T is indistinguishable from one at time T+100 milliseconds). Then, the federate may schedule events 100 milliseconds into the future, providing a lookahead of this amount.
- *Time stepped federates:* In a time stepped simulation, the lookahead is normally the size of the time step. This is because a federate 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 10 minutes. These events could therefore be scheduled immediately, resulting in a lookahead of 10 minutes.
- *Precomputing simulation activities.* If the events produced by a federate over the next L units of time do not depend on external events, but only depend on internal computations, these computations can be performed in advance, enhancing lookahead. For example, if the time until the next interaction of a federate with another federate is drawn from a random number generator, the generator can be sampled ahead of time, and the computed value can be used to derive the federate's lookahead for this time instant.

In general, lookahead may simply not be present in some federations, due to the nature of the activities being modeled. This may require the federation to reduce the fidelity of the model, i.e., introduce greater temporal uncertainty (e.g., see the third item listed above; this is not unlike increasing the size of the time step in a time-stepped simulation), or to resort to optimistic processing techniques to achieve acceptable performance.

Lookahead can change dynamically during the execution. 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 time stamp ordering) with time stamp less than C+L, where C is the federate's current 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 time stamp less than C+L are produced.

The RTI requires each federate to specify lookahead information if any events utilizing the time stamp ordered service are generated. Care must be taken in developing the federate to maximize lookahead, as this can significantly affect performance. A single lookahead value is designated by each federate. This value may change at runtime, but reductions in lookahead do not take effect immediately, as noted above.

A federate's lookahead must be strictly greater than zero to implement some of the services for advancing logical time. Specifically, a **Time Advance Grant** to logical time T indicates to the federate that all TSO messages with time stamp less than *or equal to* T have been delivered to the federate. This would be impossible to guarantee if lookahead were zero because a federate A could be given a **Time Advance Grant** to logical time T, then schedule an event also with time stamp T that is received by federate B, which in turn schedules a second event also with time stamp T that is received by A, thereby violating the guarantee that the **Time Advance Grant** had delivered all messages with time stamp T or less. The RTI will always use a small, nominal value for lookahead to circumvent situations such as this.

7. Time Management Services

HLA time management services are concerned with (1) the order that messages are delivered to federates, which in turn has implications on transportation (object management) services, and (2) the mechanisms for advancing logical time. Each of these are described next

7.1. Implications on Transportation Services

The object management services **Update Attribute Values** and **Send Interaction** are used to declare the latest state of the attributes of a federate object or parameters of a federate interaction. These "declarations" are considered to be "scheduled events" when lookahead is being exploited. Invocation of these services typically results in the generation of one or more messages transmitted to federates subscribing to the requested information.

The choices for message ordering are:

- receive order,
- priority order,
- causal order,
- causal and totally ordered, and
- time stamp order (TSO).

Categories providing increased functionality (more precise control) of delivery order will only do so at increased cost in terms of latency of message delivery and/or the network bandwidth required to support the service. Therefore, federates should always use the least costly ordering service adequate for its objectives.

7.1.1. Specification of Message Order

Different approaches may be taken regarding how the category of service is specified. For instance:

1. Each federate might specify the category of service for the messages it receives when it subscribes for that information, thereby allowing different federates to view incoming information in a way that is most appropriate for that federate.
2. Each federate may select the most appropriate category of service on each message send, thereby allowing federates within a single execution to interleave generation of messages using different categories of service in arbitrary ways. This enables federates to select the appropriate category of service based on the type of information that is being transmitted.
3. Each federate may select the category of transportation service when it announces it is publishing that information.
4. The category of service could be specified by the federate for all messages that it sends or receives.

The first approach is the most desirable from a modeling perspective because it enables each federate to treat incoming information in the manner most appropriate to the way it intends to use that information. Therefore, this is the approach recommended for the HLA. However, initial versions of the RTI will not fully implement this approach.

The current implementation of the RTI uses an approach that is a combination of the second and fourth approaches listed above. The federate generating the message specifies the message order to be used. In addition, two characteristics of a federate affect how time advance mechanisms are treated within the RTI and how the RTI deals with the federate.

- **Time Constrained**. This characteristic indicates whether the federate will be constrained by the logical time of other federates (i.e., time constrained federates may receive TSO messages).
- **Time Regulating**. This characteristic indicates whether the federate proposes to participate in determining the logical time of other federates (i.e., time regulated federates may send TSO messages).

These characteristics are independently specifiable, so that four possible types of time-evolving federates are possible:

- **Logical time synchronized**. This federate is both time constrained and time regulating. It participates in other federate time advance decisions and accepts such participation from other federates. ALSP simulations are examples of such a federate.
- **Externally time synchronized**. This federate is neither time constrained nor time regulating. It advances its internal time using mechanisms other than those provided by the RTI. DIS simulations are examples of such a federate where the wall clock is used as a synchronizing mechanism.
- **Logical time passive**. This federate is time constrained but not time regulating. It paces its internal logical time to that of the other federates but does not impact on their logical time advance decisions. Viewers and federation management tools are examples of such a federate.
- **Logical time aggressive**. This federate is time regulating but not time constrained. It impacts the time advance decisions of other federates but ignores their input to its own time advances. A script player is an example of such a federate.

Currently, the RTI treats federates with these characteristics specially, depending on whether they are time constrained and time regulating.

- **Time Constrained**. If a federate is *not* time constrained, the RTI does not provide Time Stamp Order delivery of messages. No matter how messages were specified by the originating federate, they are delivered to this federate in Receive Order. Since other federates do not impact this federate's time advance and all messages are delivered to it in Receive Order, it is not necessary for the federate to use a time advance service in order to receive messages.
- **Time Regulating**. If a federate is *not* time regulating, the RTI does not consider it when determining the LBTS (lower bound on the time stamp of TSO messages that can be received in the future) of other federates—its time is not relevant to other federates. Also, since it is not participating in the time advance of other federates, messages that this federate sends may not use the Time Stamp Order service. No matter how the messages are specified by this federate, the RTI delivers them in Receive Order.

7.1.2. Event Retraction

Event retraction refers to the ability of a federate to retract (sometimes called cancel or unschedule, but "cancel" has a different meaning here) a previously scheduled event. This is a common discrete-event simulation primitive often used to model interrupts and other preemptive behaviors. Event retraction is also utilized by optimistic federates to implement mechanisms such as "anti-messages." The **Update Attribute Values** and **Send Interaction** RTI services return a handle for the event that is used to specify the event that is to be retracted.

Event retraction is available for all categories of service. If the RTI at the destination federate receives a retraction request for an event that is not buffered in the RTI (e.g., because the corresponding message has already been forwarded to the federate or is delayed in the network), the retraction request is forwarded to the federate.

Lookahead restrictions must be applied to retracting previously scheduled events utilizing the time stamp ordered message delivery service if it is important that messages for retracted events never be passed on to other federates. Specifically, if the current time of a federate is T, and its lookahead is L, then the federate can only retract events containing time stamps greater than T+L. Messages for events containing a smaller time stamp may have already been passed to other federates.

It is possible that a message for a retracted event may be lost in the network, especially if best effort delivery is used. In this case, if the retract request was successfully delivered, the message for the retracted event will never appear, yet the retract request would still be passed to the receiving federate. Federates must be designed to allow for conditions such as this. The RTI does guarantee that if a retraction request is forwarded to a federate, the retracted message will not be later delivered to the federate.

7.2. Time Advance Services

The time advance services provide a means for the federate to control its advancement of logical time. These services are also used to control the delivery of new messages to the federate. Two services for advancing logical time are defined: **Time Advance Request** and **Next Event**

Request. It is anticipated that **Time Advance Request** will be used by time-stepped federates, and **Next Event Request** will be used by event-driven federates.

- **Time Advance Request (t):** Requests an advance of the federate's logical time to t. Invocation of this service implies that the following messages are eligible for delivery to the federate:
  (i) all incoming receive ordered messages, and
  (ii) all messages using other ordering services with time stamp less than or equal to t.
  After invoking **Time Advance Request**, the messages are passed to the federate by the RTI calling the **Instantiate Discovered Object**, **Remove Object**, **Receive Interaction** and **Reflect Attribute Values** services provided by all HLA-compliant federates. The federate may simply note the occurrence of these events for later processing, or immediately simulate actions resulting from the occurrence of the events.
  All eligible messages are delivered, and the request is always completed by the RTI calling the federate's **Time Advance Grant** service, regardless of the number of messages that are delivered. Invocation of **Time Advance Grant** indicates to the federate that no additional TSO messages with time stamp less than or equal to t will be delivered in the future.
  By invoking **Time Advance Request** with parameter t, the federate is guaranteeing that it will not generate a TSO message at any time in the future with time stamp less than t plus that federate's lookahead.
- **Next Event Request (t):** Requests the next TSO message from the RTI, provided that message has a time stamp no greater than t. Invocation of this service implies that the following messages are eligible for delivery to the federate:
  (i) all receive ordered messages,
  (ii) the smallest time stamped TSO message that will ever be delivered in the future with time stamp less than or equal to t, and all other TSO messages containing this same time stamp value, and
  (iii) all messages using other ordering services with time stamp less than or equal to t.
  These messages are delivered to the federate by the RTI calling the **Instantiate Discovered Object**, **Remove Object**, **Receive Interaction** or **Reflect Attribute Values** services provided by the federate. A **Time Advance Grant** completes this request and indicates to the federate that it has advanced its logical time to the time stamp of the TSO message(s) that is (are) delivered, if any, or to time t if no TSO messages were delivered.
  By invoking **Next Event Request** with parameter t, the federate is guaranteeing that if it does not receive any additional TSO messages in the future with time stamp less than t, the federate will not later generate any TSO messages with time stamp less than t plus the federate's lookahead.
- **Request Federate Time:** Requests the current value of federate time (defined in Section 3).

After a **Time Advance Request** or **Next Event Request** service request has been completed and all message delivery and/or **Time Advance Grant** service invocations have been performed, the RTI will not deliver additional messages to the federate until the next **Time Advance Request** or **Next Event Request** service invocation. Federates may not initiate a new **Time Advance Request** or **Next Event Request** service if the previous one has not yet completed.

The following services provided by the federate are invoked by the RTI:

- **Time Advance Grant:** Invocation of this service indicates a prior request to advance logical time has been honored. Specifically, invocation of this service indicates:
    - if the grant is in response to a **Time Advance Request** invoked by the federate, the logical time of the federate is that specified in the **Time Advance Request** and the RTI has delivered all TSO messages to the federate with time stamp less than or equal to the time specified in the **Time Advance Request**.
    - if the grant is in response to a **Next Event Request** invoked by the federate, the logical time of the federate is that of the TSO message(s) delivered to the federate if any were delivered (all such messages must have the same time stamp), or the time specified in the **Next Event Request** if no TSO messages were delivered. Once the grant has been received, no subsequent TSO message will be delivered to the federate with time stamp less than or equal to the federate's logical time.

  In all cases, **Time Advance Grant** informs the federate that all TSO messages with time stamp less than or equal to the federate's current logical time have been delivered to the federate. This property enables the federate to ensure that all event notices (both internal to the federate and generated by other federates) are processed in time stamp order, if this is a requirement for correct execution of the federate.
- **Instantiate Discovered Object**, **Remove Object**, **Receive Interaction** and **Reflect Attribute Values:** These are object management services used to cause the delivery of messages to the federate.

Because the **Time Advance Grant** call (or any action that advances logical time) will only be made after the RTI can guarantee that no future messages will arrive with time stamp less than or equal to the federate's logical time, this call cannot be made until the RTI can make such a guarantee. Similarly, message deliveries may be delayed after the time advance service has been invoked by the federate until the RTI can guarantee time stamp ordered message delivery. The delays that are introduced may be significant, and depend on the RTI's synchronization protocol that is responsible for guaranteeing time stamp ordered delivery, and the amount of lookahead in the federation. This phenomenon is commonly referred to as "artificial blocking" in the parallel simulation literature.

It is noteworthy that as defined above, **Time Advance Request**, **Next Event Request**, and **Time Advance Grant** only pertain to the advancement of logical time. Wallclock time advances independent to the federate's actions (of course!).

8. Example Usage

Typical examples of using the time management services for federates using different internal time flow mechanisms are described next. These examples should *not* be interpreted as the only way to use the HLA-TM services, but only as possible ways the services might be used.

8.1. Time-Stepped (Periodic) Federate

A time-stepped federate operates synchronously, with the federate advancing from one time step to the next after work in the current time step has been completed. Computations corresponding to a time step can only schedule new events into the next (or some future) time step, but not the current one. A typical time-stepped federate might use the HLA-TM services as described below. In this example, delivered messages update local data structures within the federate, and

the new state of the federate is computed and new events are scheduled after the **Time Advance Grant** is received.

```
/* now is a local variable tracking the logical time of the federate */
while (simulation execution still in progress)
    compute state of federate at time now
    provide any changed information (new attribute values or interactions) to the RTI via
        the Update Attribute Values and/or Send Interaction services with time stamp
        now+timeStepSize  (or larger) for the next (or future) time step.
    /* receive all external event notices in next time step: (now, now+timeStepSize]  */
    invoke Time Advance Request(now + timeStepSize)
    honor zero or more RTI requests for Reflect Attribute Value and Receive
        Interaction services
    honor RTI service request for Time Advance Grant
    now = now+timeStepSize
```

A scaled real-time version of the time-stepped federate can be realized by (1) implementing an algorithm in each federate to coordinate logical time with scaled wallclock time or (2) creating a "pacing" federate whose only task is to coordinate logical time with scaled wallclock time. Using this second mechanism permits the other federates to operate as fast as possible or paced to wallclock time depending on the presence of the pacing federate.

To implement the time stamp ordered delivery service, the RTI must compute a value called LBTS (lower bound on time stamp) for each federate. The LBTS indicates a lower bound on the time stamp of any subsequent message the RTI at a particular federate will receive from another federate. The RTI may only release TSO messages to a federate that have a time stamp less than LBTS because a message with a larger time stamp may be preceded (in time stamp order) by another message that has not yet been received. Ignoring messages "in transit," the LBTS for a federate F can be computed as $\min(T_i+L_i)$ over all federates i that can send F a TSO message, where T_i and L_i are the logical time and lookahead, respectively, of federate i.

Figure 3 (a) shows a typical scenario, depicting interactions between a time-stepped federate and the RTI. This example assumes a real-time execution with scale factor of 1, and uses a time step size of five units of time. All messages are assumed to be TSO. This figure shows the invocations of services between the federate and RTI, as well as important clock variables.

A "clock" federate (not shown in the figure) is used to pace the execution with wallclock time. The clock federate advances in logical time in synchrony with wallclock time, i.e., the logical time of the clock federate exactly matches wallclock time. The clock federate is assumed to have a lookahead of 5. Thus, the LBTS (and the logical time) of any federate can never exceed wallclock time by more than 5 units of time. To simplify the example, the time required by the synchronization protocol to update LBTS values is assumed to be negligible, as is clock skew for the real-time clocks, i.e., at any instant, all federates observe the same value of wallclock time.

The scenario begins at the top of the time-stepped loop, with the federate computing its new state corresponding to time 35, and sending a new attribute value (by invoking the **Update**

Attribute Values service) with a time stamp of 40, the time of the next time step. At this point, the federate is "ahead" in the simulation because it is at logical time 35, and wallclock time is only 32. A **Time Advance Request** invocation to the RTI is made, indicating the federate is ready to advance to the next time step. This signals the RTI to deliver all messages with time stamp of 40 or less to the federate. The RTI has two messages waiting to be delivered, an attribute update with time stamp 39, and an interaction with time stamp 40. However, the RTI must first wait until it can determine that no other TSO messages will be delivered with time stamp smaller than these in order to guarantee time stamp ordered delivery. This guarantee comes at wallclock time 36 when LBTS advances from 36 to 41. This change in LBTS will be elaborated upon later. The two messages are now delivered to the federate by the RTI invoking the federate's **Reflect Attribute Values** and **Receive Interaction** services, and the federate's logical time is advanced to 39, and then 40.

Because LBTS is 41, the RTI can guarantee this is all of the messages that will be delivered with time stamp 40 or less, so the RTI informs the federate of this fact by invoking the federate's **Time Advance Grant** service. The federate can now proceed with the next time step, i.e., compute the state of the federate at time 40. The LBTS for this federate now advances in synchrony with wallclock time (plus 5, the clock federate's lookahead) because in this scenario, no other federates send it a TSO messages with time stamp less than 44, so LBTS is determined entirely by the clock federate.

8.2. Event-Driven Federate

An event driven federate processes event notices one after the other, in time stamp order. In this example, when a TSO message or a local event notice is processed, the logical clock of the federate is advanced to the time stamp of that message/event notice. The federate must interleave execution of local event notices with messages delivered by the RTI so that all are processed in time stamp order. In this example, all messages are processed as they are delivered by the RTI. The **Reflect Attribute Values** and **Receive Interaction** procedures act as "event handlers" that update the state of the federate, and schedule new events.

```
/* 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
    /* next statement enables delivery of next external messages */
    invoke Next Event Request (TSlocal) service
        Honor zero or more RTI requests for Reflect Attribute Values and Receive
        Interaction services, providing any changed information (new attribute values or
        interactions) to the RTI via the Update Attribute Values and/or Send Interaction
        services.
    honor RTI service request for Time Advance Grant
    if (no TSO message(s) received in above RTI service requests)
        now = TSlocal
        process the next local event notice identified above, providing any changed
        information (new attribute values or interactions) to the RTI via the Update
Attribute
        Values and/or Send Interaction services.
```

```
    else
        now = time stamp of last TSO message delivered to federate
```

Like the time-stepped program, this program may also be used in a real-time federation by implementing an algorithm to coordinate logical time to scaled wallclock time or by using a pacing federate.

A scenario depicting the execution of an event driven federate is depicted in Figure 3(b). Like the previous example, execution is paced with wallclock time by a clock federate with lookahead of 5. This scenario begins with the federate "behind" in the execution at logical time 30, e.g., processing an event notice with time stamp 30. At this point, the federate is processing a local event notice, so execution is at the "then clause" of the "if" statement in the sample event driven program given above. The federate invokes the **Send Interaction** service, resulting in a message being sent with time stamp 40. As shown in the sample event driven program, execution now moves to the top of the loop, where the federate checks for local event notices *within itself*. Suppose it observes that the smallest time stamp value among its local event notices is 42. The federate invokes **Next Event Request** specifying 42 as the time stamp value, i.e., it requests the smallest time stamped message from the RTI with time stamp less than or equal to 42.

At this point, the RTI observes that it has received and buffered a message with time stamp 40. However, the RTI must delay delivery of this message until LBTS advances beyond 40, or there is danger of receiving a message with a smaller time stamp. In this example, the federate is behind all others, so LBTS advances are determined entirely by the clock federate, and its lookahead of 5. LBTS advances to 41 at wallclock time 36 because the clock federate is at logical time 36. At this point, the RTI (i) advances the federate's logical time to 40, the time stamp of the next message, (ii) delivers the time stamp 40 message to the federate by invoking the federate's **Reflect Attribute Values** service, and (iii) invokes the **Time Advance Grant** service. The federate processes the time stamp 40 message, and at wallclock time 39, reissues the **Next Event Request** with time value of 42, because the local event notice at time 42 remains. At wallclock time 39 the **Time Advance Grant** is issued, indicating there are no messages with time stamp 42 or less, so logical time is advanced to 42, and the federate may now process the local event notice with time stamp 42.

The observant reader will notice that the message sends depicted in Figure 3(a) correspond, at least in part, to the received messages in Figure 3(b) and vice versa. For example, the attribute value sent by the time-stepped federate at wallclock time 32 corresponds to the attribute value received by the event driven federate at wallclock time 36. These two scenarios, in fact, depict the concurrent execution of a federation consisting of an event driven and a time stepped federate.

This example illustrates the synchronization mechanism between the two federates to ensure messages are delivered in time stamp order. Initially, the LBTS of the time-stepped federate is 36 because the event driven federate is at logical time 30, and has a lookahead of 6. The LBTS of the event driven federate is determined by the clock federate throughout this scenario (i.e., it is wallclock time plus 5, the clock federate's lookahead) because the time-stepped federate is sufficiently far ahead in the simulation (early in the scenario, the time-stepped federate requests

a time advance to 40, guaranteeing that it will not send any new messages with time stamp less than 45). When the LBTS of the event driven federate advances to 41 at wallclock time 36, it can advance its logical time to 40. At this point, the event-driven federate no longer "holds back" the time-stepped federate because the smallest time stamped message it can send will have a time stamp of at least 46. The LBTS in the time-stepped federate is now determined by the clock federate, so the LBTS in the time-stepped federate advances to 41 (wallclock time plus 5). This allows the time-stepped federate to proceed to the next time step.

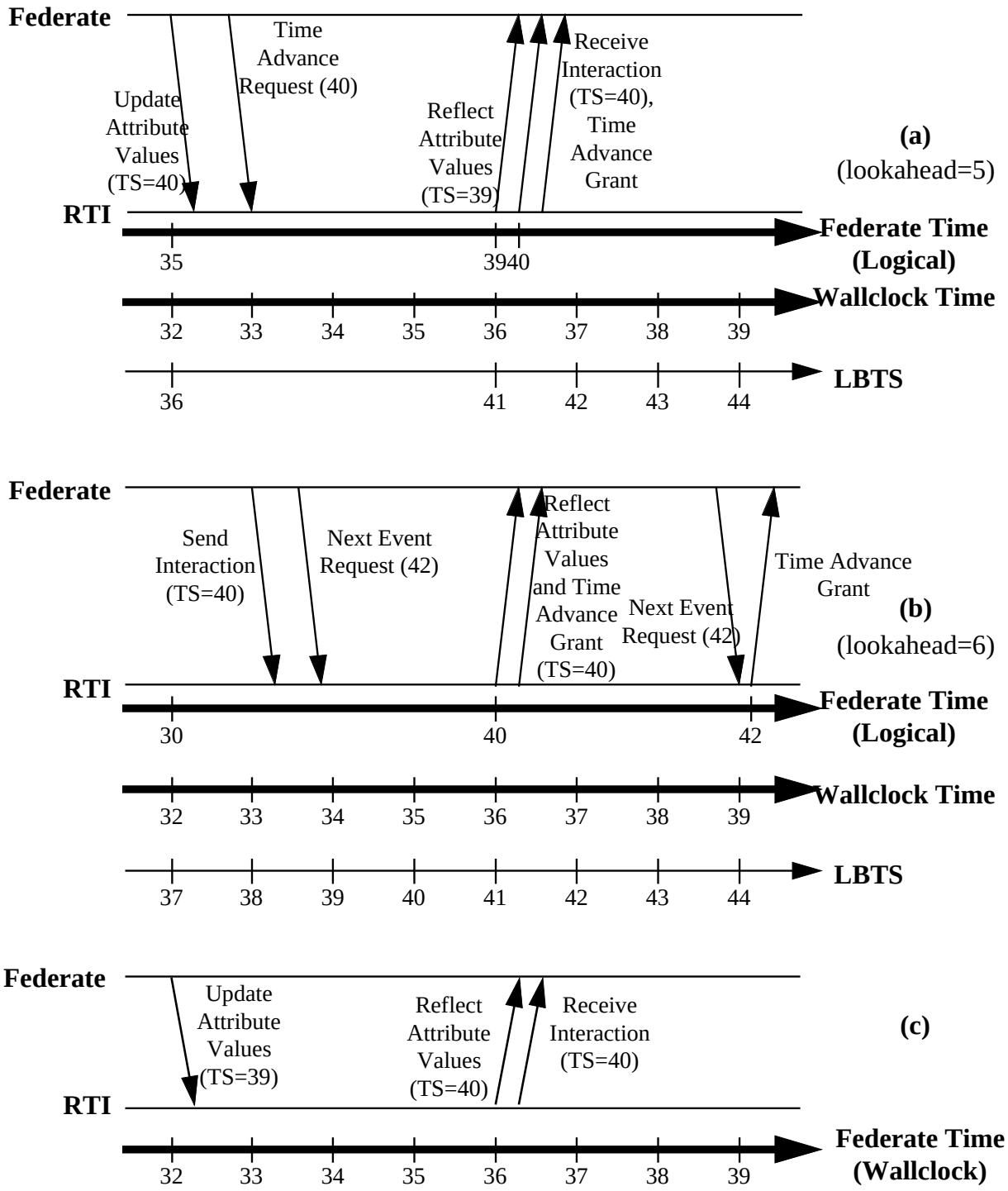

Figure 3. Execution scenarios. (a) Time stepped federate using Time Advance Request. (b) Event driven federate using Next Event Request. (c) Independent time advance federate.

.

8.3. Independent Time Advance Federate

In its simplest form, an independent time advance federate is one that does not require coordination with other federates to advance through time. Such federates processes messages either as they arrive, or according to the time stamp value and its relationship to scaled wallclock time. DIS simulations often utilize this mode of operation.

Establish federate as unconstrained by the logical time of other federates and unregulating of the logical time of them.
While the federate is executing
    Determine local time
    Invoke **Update Attribute Values** and/or **Send Interaction** services when declaring a new state of an object or when declaring an object interaction to the federation via the RTI; assign a time stamp based on local time.
    Honor RTI requests for **Reflect Attribute Values** and **Receive Interaction** services.

A scenario depicting the execution of an independent time federate that utilizes time stamp ordered message services is depicted in Figure 3(c). This federate generated the attribute value update (time stamp 39) received by the time-stepped federate in Figure 3(a) and receives both of the messages sent by the time-stepped and event driven federates. At wallclock time 32, it performs an attribute update to which it assigns a time stamp 39 that is sent to the time-stepped federate. Shortly thereafter (at approximately wallclock time 33 or 34) it receives two messages, both with time stamp 40, with one coming from each of the other federates. Since the federate is unconstrained by the logical time of other federates, the RTI delivers these messages immediately. The independent time federate must determine how to process these "future" events.

An independent time federate does not affect the LBTS of other federates; therefore it does not impede their progress in advancing logical time.

In summary, Figure 3 illustrates interoperability among time stepped, event driven, and independent time advance federates, the major categories that must be supported by the HLA. Perhaps noteworthy aspects of this scenario are the way LBTS is used to provide message ordering, the use of lookahead to (hopefully) deliver messages to federates before the time that they take affect, and the way execution is paced in the coordinated time advance federates. This latter point is elaborated upon in greater detail next.

9. Real-Time (Constrained) Simulations

The relationship of the operation of the RTI to real time executions merits closer examination. Recall that the current time of the federate is defined as one of two quantities:

- an internally calculated time (such as scaled wallclock time), or

- the logical time of the federate (advanced by the **Time Advance Request** and **Next Event Request** services).

As-fast-as-possible simulations used in a non-real-time federation execution will use logical time only. Real-time federates that do not require coordinated time advance with other federates, e.g., federates using only real-time synchronization as in DIS, do not use the logical time clock and coordinate internally to wallclock time. The case of real-time federates requiring coordinated time advances (e.g., as-fast-as-possible distributed simulations operating in a real-time setting) is discussed next.

Real-time (constrained) federates with coordinated time advances may use the RTI based on the following paradigm:

- A "timing federate" is used that makes logical time advance requests in synchrony with wallclock time advances, e.g., the federate might request a time advance request of one second for each second that wallclock time advances. Suppose the timing federate's logical time advances in synchrony with wallclock time. If the lookahead of the timing federate is L_T, the federation will be paced in the sense that no federate will be able to advance its logical time more than L_T units of time ahead of wallclock time (in the timing federate), and no TSO message will be delivered with time stamp larger than wallclock time plus L_T.
- The time stamp on each event denotes a deadline by which processing of the message containing information concerning the event should be completed. If messages are received at the destination RTI before their time stamp, they can be delivered to the federate once wallclock time (plus communication delays to advance LBTS) has advanced within L_T of the message time stamp. The federate may process the new message immediately, or it may chose to postpone processing the message if it is still "too early." The latter situation might arise if L_T is set to a large value.
- Federates attempt to "stay ahead" of real-time by scheduling events sufficiently far "into the future" (time stamp larger than the current time) so that they can be transmitted, delivered to receiving federates, and processed before their deadline has passed.
- Logical time (controlled by the federate generated time advance requests and RTI generated grants) controls delivery of all except receive ordered messages to the federate. Ideally, each federate can issue logical time advance request early enough that it can remain ahead of wallclock time, causing the federate's time advances to be paced by wallclock time.
- If logical time does *not* keep up with wallclock time, either because of delays in the RTI, federate, or the communication network, one or more federates will fall "behind" relative to wallclock time. The RTI will delay delivery of TSO messages in order to ensure message ordering guarantees are met. This will, in turn, slow the granting of time advances of other federates, and could eventually cause many federates using coordinated time advance to proceed slower than real-time. In this case, the federation is paced by logical time advances, which are advancing slower than real-time. This will be problematic for certain federations. To address this problem, federates may "turn off" time management services, effectively converting all TSO ordered messages to receive ordered messages. This prevents one federate from delaying the progress of the rest of the federation. Federations should be designed, however, so that slow federates adapt to increase their rate of time advances.

In general, it is clear that if an message is not delivered to the federate prior to its time stamp, it will be impossible to process it before its deadline. Thus certain real-time constraints must be

met for the federation to operate in real-time. A necessary, though not sufficient constraint for real-time execution without missed deadlines is:

E_{TS} > T(real[sender]) + WC(max-error) + Comm(max-latency) + HLA-TM-delay

where:

- E_{TS} is the time stamp assigned to the event.
- T(real[sender]) is the scaled wallclock time when the sending federate invoked the **Update Attribute Values** or **Send Interaction** service to pass the message to the RTI.
- WC(max-error) is the maximum difference between scaled wallclocks in the federation, e.g., due to skew in the hardware clocks of different federates. This error may be negligible in some federations.
- Comm(max-latency) is the maximum latency incurred in the communication network multiplied by the scale factor.
- HLA-TM-delay is the time the RTI must hold the message before delivering it to the federate multiplied by the scale factor. This includes other time-overheads associated with delivering a message to the application as well as the time the RTI must hold the message until ordering can be guaranteed, for TSO messages.

In short, the federate needs to send the message far enough into the future so it can be received before its time stamp.

10. Optimistic Time Management Services

The discussion thus far has focused entirely on conservative synchronization that avoids the possibility of processing messages out of time stamp order. The other well-known approach to synchronization are so-called *optimistic* techniques that allow messages to be processed out of time stamp order, but use a rollback mechanism to recover. The HLA-TM services described next enable optimistic federates to utilize HLA-TM services while still enjoying the advantages afforded by optimistic execution.

The optimistic HLA-TM services describe below *do not require all federates to support a rollback and recovery capability*. 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.

Several additions are used to support optimistic execution, as described below. The discussion that follows only pertains to TSO messages.

1. Federates may 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.

2. Because the federate optimistically receives messages, an RTI primitive is required for the federate to indicate to the RTI its current logical time, as discussed below.
3. The LBTS value must be available to the optimistic federate.

The following optimistic time management services provide this functionality:

1. **Flush Queue Request (t):** This primitive releases *all* messages stored in the RTI's internal queues and delivers them to the federate invoking this service. Time stamp ordered messages are delivered, despite the fact that the RTI may not be able to guarantee that messages containing a smaller time stamp could arrive later. The parameter t indicates that if the federate does not receive any additional time stamp ordered messages with time stamp less than t, then the federate's logical time may be advanced up to t. By invoking this service, the federate indicates it will not generate any new time stamp ordered messages with time stamp less than t plus the federate's lookahead if it does not receive any new time stamp ordered messages with time stamp less than t, similar to the time parameter in the **Next Event Request** service. A federate that includes a Time Warp simulation will typically specify the smallest time stamped unprocessed event stored within the federate as the parameter to this call. This service request is completed by the RTI invoking the federate's **Flush Queue Grant** service.
2. **Flush Queue Grant (t):** Invocation of this service indicates the **Flush Queue Request** service has completed. The time parameter of this call indicates that logical time for the federate has been advance to this value, and no additional time stamp ordered messages will be delivered in the future with a time stamp less than this value. This time parameter will be the lesser of LBTS and the time parameter of the **Flush Queue Request** that resulted in this call.

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 optimistic message facility to receive, and optimistically process event notices.
- Optimistically generated messages are transmitted through the RTI to other federates, the same as ordinary, non-optimistic messages. The RTI does not distinguish between optimistic and conservative 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, annihilation happens 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 conservative synchronization mechanism to prevent conservative federates from receiving optimistic messages. 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:

```
Initialize RTI /* LogicalTime := 0.0, LBTS := min(Lookahead_i) for all i */
GVT := 0.0
FlushQueueRequest( min time stamp among local events )
while GVT < FederateEndTime
    NextEventTS := min time stamp among local events
    if TimeAdvanceGrant(t) has not been invoked
        Allow RTI to deliver any outstanding events
        Add non-retraction events to message list
        Add retraction events to retraction list
    else
        GVT := t
        FossilCollect( GVT )
        FlushQueueRequest( NextEventTS )
    while message list is not empty
        if TS head of message list < TS of last processed event for federate
            Rollback( head of message list ) /* May cause retraction events to be sent */
        Enqueue head of message list into federate's local event queue
        Remove head of message list
    while retraction list is not empty
        find retractionlist head in unprocessed or processed message queue of federate
        if head of retraction list has been processed
            Rollback( head of retraction list ) /* May cause retraction events to be sent */
        delete head of retraction list from Federate
    Dequeue next event to be processed /* May not be the "original" next event from above
*/
    Save state
    Process event
```

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.

One limitation of the above approach is it assumes a receive time stamp based definition of GVT. Some memory management protocols (e.g., Cancelback, message sendback) require a somewhat different definition of GVT. Some modifications to the above mechanism are required to support this alternate definition of GVT for optimistic federates using such techniques.

11. Interactions Between Time and Declaration Management

Declaration management mechanisms (also referred to as filtering) control which federates receive messages for each attribute update and interaction. For example, a federate modeling a tank might wish to receive position updates for all vehicles within a specified geographic sector. The set of federates that should receive updates to a particular attribute will change dynamically during the federation execution, e.g., the tank in the previous example should begin to receive updates to another vehicle's position attribute if and when that vehicle moves into the specified sector, and stop receiving updates when the vehicle moves out of the sector. The RTI must maintain a database so that it can determine which federates should receive messages for each attribute update and send interaction request. In some federations it is important that changes to this database be properly synchronized with attribute updates and interactions so that each federate receives messages corresponding to all information to which it has subscribed, and ideally, no others. This issue is of particular importance to federates that require time stamp ordered event processing and coordinated time advances.

In independent time advance federates paced by wallclock time, the semantics of subscribe and unsubscribe operations is based on message arrival time: the subscription (unsubscription) takes effect as soon as the service is invoked, ignoring the time for the RTI to perform the operation. Coordinated time simulations utilizing logical time usually require different semantics, however. The temporal semantics of subscription and unsubscription operations should take effect at specific points in *logical time.* If a radar federate subscribes to receive attribute updates at time T because a vehicle moves into its range at time T, it expects to receive all updates with time stamp greater than or equal to T.

Ensuring that each federate receives all of the messages to which it has subscribed based on *logical time* semantics is a non-trivial matter. If a federate subscribes to an attribute value at federate time T_1, and unsubscribes at time T_2, then it should receive all attribute updates with time stamp in the interval $[T_1, T_2)$. To adhere to lookahead constraints, federates will generate attribute updates and interactions “in the future,” i.e., with time stamps larger than the federate’s current time. If the data distribution mechanism simply delivers messages to those federates subscribed to the information when the **Update Attribute Values** or **Send Interaction** service is invoked, a federate may *not* receive all of the messages to which it has subscribed. Consider the following scenario. A federate performs an attribute update with time stamp 100. The RTI transmits messages to the federates subscribed to receive this information. After the messages have been transmitted, another federate now subscribes to this attribute at time 95, and does not unsubscribe. This federate should have received the update message at time 100. The RTI must

somehow ensure that the federate receives the update at time 100, even though the messages for that event may have already been transmitted to other federates. This implies that the RTI must do more than simply send messages to the federates subscribed to an attribute when the **Update Attribute Values** or **Send Interaction** service is invoked. This section describes the semantics of logical time-based subscription and unsubscription operations, and describes an approach to avoiding errors such as that described above.

11.1. Declaration Management

It is convenient to view declaration management as logically being composed of two layers of software (see Figure 4). The upper layer provides a convenient interface to the federate for specifying its "interests" via certain interest expressions, e.g., based on filter spaces. This interest management software receives and processes interest expressions produced by federates and generates for the lower layer *Add* and *Delete* operations to change the database indicating which federates receive attribute updates and interactions. At this lower layer, distribution list software performs changes to the database and ensures that these changes are properly synchronized with the attribute updates and interactions so that each federate receives all of the messages it is supposed to receive, and no others.

Complete specification of the interest management services is beyond the scope of this document. The remainder of this section focuses on synchronization issues in the distribution list software. This discussion deals exclusively with time stamp ordered messages, and the mechanisms are primarily of interest to federates that require time stamp ordered processing of events.

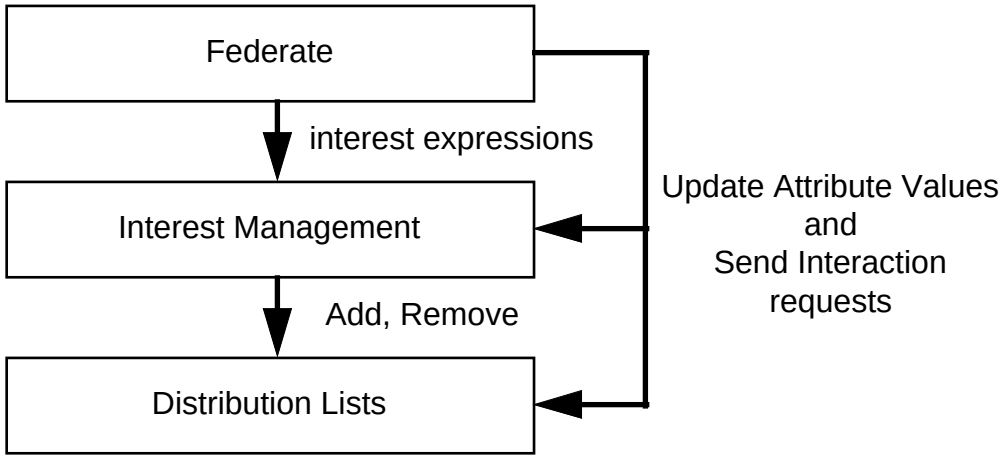

Figure 4. Logical architecture of declaration management functions.

11.2. Problem Statement

Logically, the RTI can be viewed as maintaining a *distribution list* for each object attribute indicating which federates should receive updates to that attribute. The distribution lists collectively form the database mentioned earlier. This concept can be easily extended to

interactions, so here, we discuss only attribute updates. Each change in a distribution list has a time stamp associated with it denoting a point on the federation time axis when the change occurs. Thus, one may conceptually view each distribution list as evolving, one change at a time, over successive points on the federation time axis.

D(A,T) denotes the distribution list for attribute A corresponding to federate time T. If an update to A occurs with time stamp T, the RTI will send a message to each federate in D(A,T). The following operations may be performed on the distribution list for A (see Figure 4):

- **Add(F,A,T):** add federate F to the distribution list of attribute A to take effect at federate time T (e.g., a federate has entered the radar range of another federate, so the latter should begin receiving state updates of the former).
- **Delete(F,A,T):** remove federate F from the distribution list of attribute A as of federate time T (e.g., the radar for F can no longer detect another federate X because at time T, X is too far away, so F should no longer receives state updates for X).
- **Update(A, V, T):** update attribute A at time T with value V. This primitive would be invoked when the federate invokes the **Update Attributed Values** (or **Send Interaction**) service.

The Add (Delete) operation causes the federate to subscribe (unsubscribe) to receive updates to attribute A. These operations are invoked *within* the RTI, and are not directly accessible by the federate. They may be invoked when a change in the filter specification occurs that changes the set of federates that should receive messages, when an object becomes "discovered" (or removed), or when a new object is instantiated (or removed). The semantics of these operations are defined as follows:

- Composing the operations Add(F,A,T) and Delete(F,A,T) with the same parameter values has the same effect as if neither operation were performed. In this case, the two operations are said to be *canceled.*
- Add (F,A,T): let T_D be the smallest time stamped Delete operation (ignoring canceled operations) by federate F on attribute A such that $T_D > T$. F will receive a message for every update to attribute A with time stamp in the interval $[T, T_D)$.
- Delete (F, A, T): let T_A be the smallest time stamped Add operation (ignoring canceled operations) by federate F on attribute A such that $T_A > T$. F will *not* receive any messages for updates to attribute A with time stamp in the interval $[T, T_A)$.

These primitives assume federates should *only* receive messages for information it had subscribe to, and no others. This requirement may sometimes be relaxed in practice because the federate may simply ignore messages carrying information it had not requested.

11.3. A Synchronized Data Distribution Mechanism

We now describe a mechanism to implement the Add, Delete, and Update operations described above. If Add, Delete, and Update operations for each attribute were issued sequentially to the RTI in time stamp order, data distribution would be trivial. In that case, Add and Delete operations would simply update the distribution list, and Update operations would transmit messages to the destinations in the list. Because the operations do not arrive in time stamp order, the RTI must maintain different versions of the distribution list corresponding to different points on the federation time axis. An Update operation with time stamp parameter T reads the

version of the distribution list corresponding to time T, and Add or Delete operations at time T create a new version of the list effective at time T.

The distribution list for attribute A corresponding to time T is constructed by determining which federates have subscribed via the Add operation to receive updates with time stamp T.

Specifically, define the "subscription function" as follows:

S(F, A, T)

= TRUE if an uncanceled operation Add (F, A, T_A) exists such that $T_A \leq T$ and no uncanceled operation Delete (F, A, T_R) exists such that $T_A < T_R < T$.

= FALSE otherwise

D(A,T) is defined as the set of federates F_i such that $S(F_i, A, T)$ is true.

The scenario described earlier illustrated a situation where a federate subscribes to an attribute after an update to that attribute has already been transmitted to other federates. This suggests that update attribute messages must be logged by the RTI so that subscription requests that later arrive can cause these messages to be resent. A log L(A) for attribute A is defined for this purpose. This log is defined as a sequence of tuples $<V_i, T_i>$ where V_i is the new value contained in the update message sent for an attribute update with time stamp T_i. The log only contains copies of messages transmitted to the RTI (not unlike message buffers elsewhere in the RTI) and does not interpret this state information, so this does not violate the HLA principle that the RTI does not store the federate's state.

Consider the sequence of Add and Delete operations performed by a *single* federate on a *single* attribute. In general, these operations need not arrive at the processor managing the distribution list for that attribute in time stamp order. This is because the federate may not issue these operations in time stamp order, or even if it did, the relative order of the operations may not be preserved as the associated messages are transmitted through the network. Thus, at any instant, there need not necessarily be a perfect pairing of Add and Delete operations. For example, Figure 5 shows a situation that could arise where a Delete operation with time stamp 20 has been delayed in the network, giving the temporary appearance that there is an extra Add operation. In this snapshot (i.e., prior to receiving the time stamp 20 Delete operation), the extra Add operation at time 25 is, in effect, a null operation because the federate already subscribed to the attribute at time 15. Because Add and Delete operations with the same parameters cancel, the federate is subscribed to receive updates containing a time stamp in the intervals [5,10) and [15,35).

It is convenient to view the history of Add/Delete operations to an attribute by a federate according to a diagram such as that shown in Figure 5. Consider a new Add or Delete operation with time stamp T. The status of the federate (subscribed or unsubscribed) at time T only depends on the largest time stamped uncanceled Add or Delete operation with time stamp smaller than T. For example, in Figure 5, when the Delete operation with time stamp 20 arrives, the Add operation at time 15 indicates the federate is subscribed to the attribute at time 20, independent of what other Add or Delete operations occurred with time stamps less than 15. Similarly, the effect of the new operation at time 20 only persists until the next higher time

stamped uncanceled Add or Delete operation. In Figure 5, the effect of the new operation at time stamp 20 only persists until time 25. No operation with time stamp larger than 25 is affected by this new operation.

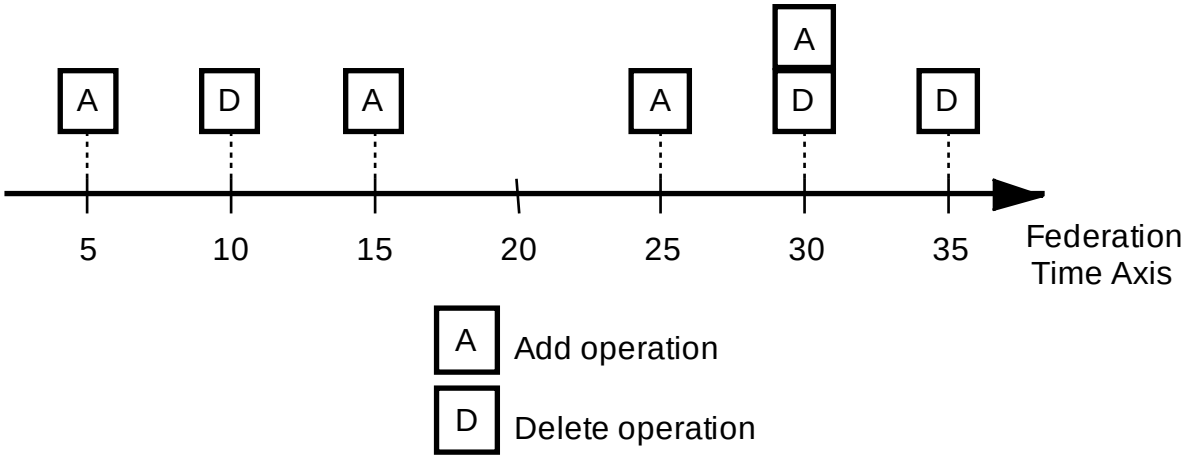

Figure 5. Snapshot of Add and Delete operations by a single federate. The federate is subscribed to receive updates in the intervals [5,10) and [15,35). the two operations at time 30 are canceled.

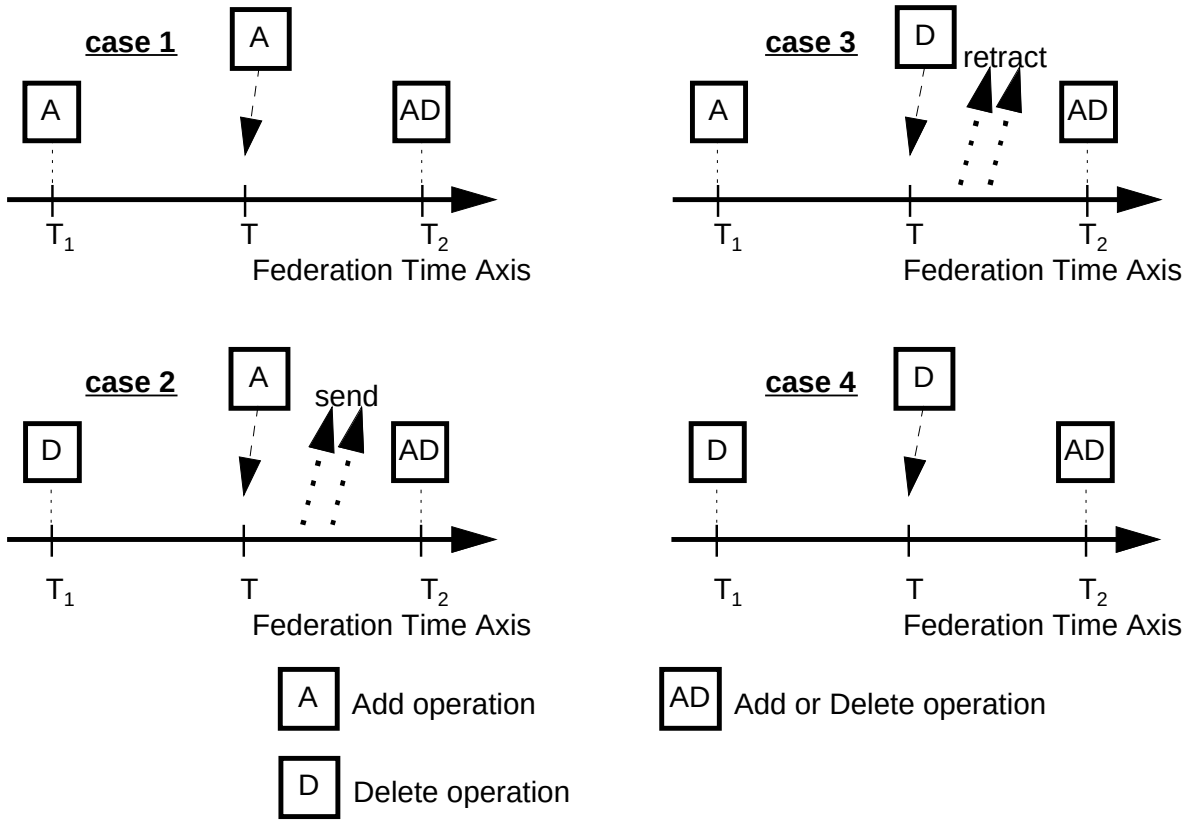

Figure 6. Possible cases when a new Add or Delete operation arrives.

Based on the above observations, one can see that when a new Add or Delete operation arrives with time stamp T, the "effect" of the operation only spans the interval from the time stamp of the immediately preceding (in time stamp order) Add/Delete operation to the immediately following operation. In the discussion that follows, only uncanceled Add and Delete operations are considered. One can enumerate all possible situations. There are eight possible combinations. The first four cases correspond to a newly arriving Add operation. It can be preceded (followed) by an Add or a Delete operation (see cases 1 and 2 in Figure 6). Actually, of these four cases, only two are different because the processing of the new operation does not depend on whether an Add or Delete operation follows. Case 1 in Figure 6 corresponds to a new Add operation preceded by another Add operation, and case 2 corresponds to the new Add preceded by a Delete operation. Similarly, there are effectively two cases to consider when a new Delete operation arrives: it may be preceded by an Add operation (case 3 in Figure 6) or it may be preceded by a Delete operation (case 4 in Figure 6). To account for the smallest and largest time stamped Add/Delete operations (which do not have a preceding or following operation, respectively), we implicitly assume there are Delete operations at times $-\infty$ and $+\infty$ to represent the fact that the federate is not subscribed to any attribute initially, nor after the end of the execution.

Correct realization of the Add and Delete operations can be derived from the case analysis shown in Figure 6. First consider Add operations. Case 1 corresponds to a new Add operation where the federate is already subscribed to the attribute. This situation of two consecutive (in

federate time) Add operations would arise if the Delete operation for the earlier Add operation had been delayed. The new Add operation is not unlike that at time stamp 25 in Figure 6, i.e., this operation has no effect because the federate is already subscribed to the attribute. Therefore, no further action is required other than noting that the Add operation has occurred. Case 2 corresponds to the situation where the federate is not subscribed to the attribute. The federate should, but has not yet received any attribute updates with time stamp in the interval [T, T_2), where T_2 is the time stamp of the following Add/Delete operation, so messages for these updates must be sent to the federate. Updates with time stamp larger than T_2 have already been correctly processed by the operation at time T_2.

Now consider Delete operations. Case 3 corresponds to a Delete operation when the federate is subscribed to the attribute. In this case, the federate has been sent messages with time stamp in the interval [T, T_2), where T_2 is the time stamp of the following Add/Delete operation, so these messages must be retracted (canceled). The retractions are unnecessary if it is permissible to receive additional messages beyond what the federate had subscribed to receive. Case 4 corresponds to a Delete operation occurring when the federate is not subscribed to receive updates. Like case 1 for the Add operation, no additional messages (or retractions) need to be sent.

Update operations do not directly affect the status (subscribed or unsubscribed) of a federate, so do not impact the Add or Delete operations. If the update causes a change in the attributes to which a federate is subscribed (e.g., an attribute update might indicate a vehicle is no longer visible), a separate Add or Delete operation must be generated.

Based on this case analysis, the Add, Delete, and Update operations can be realized as follows:

I. Update (A,V,T):
   A. send V to all federates in D(A,T)
   B. record <V,T> in L(A)

II. Add(F,A,T):
   A. record that F has been added to D(A) at time T
   B. if not S(F,A,T) then /* case 2, not subscribed to receive updates */
      1. let AD be the smallest time stamped uncanceled Add or Delete operation for F on A with time stamp greater than T, and let T_2 be the time stamp of AD.
      2. For each tuple <V,T_v> in L(A) where $T \leq T_v < T_2$, send V to F

III. Delete (F,A,T):
   A. record that F has been deleted from D(A) at time T
   B. if S(F,A,T) then /* case 3, subscribed to receive updates */
      1. let AD be the smallest time stamped Add or Delete operation for F on A with time stamp greater than T, and let T_2 be the time stamp of AD.
      2. For each tuple <V,T_v> in L(A) where $T \leq T_v < T_2$, send a retraction of V to F

The Add and Delete operations must be recorded so that D(A,T) can be properly computed.

In a distributed implementation, the distribution list and value log for an attribute could be implemented in a single processor, e.g., the processor containing the federate that owns the

attribute. The distribution lists and logs for different attributes may reside in different processors. Add and Delete operations require that a message be sent to the processor containing the distribute list and log for the attribute specified in the Add/Delete operation.

11.4. Correctness

The correctness of the synchronization mechanism described above is based on maintaining the following properties:

1. if a federate is added to a distribution list by an Add operation with time stamp T_1, and the smallest time stamped uncanceled Delete operation with time stamp greater than T_1 has a time stamp of T_2, then a message should be sent to the federate for each attribute update with time stamp in the interval $[T_1, T_2)$.
2. if a federate is removed from a distribution list by a Delete operation with time stamp T_1, and the smallest time stamped Add operation with time stamp greater than T_1 has a time stamp of T_2 (T_2 is $+\infty$ if no such Add operation exists) then no messages will be sent to the federate with time stamp in the interval $[T_1, T_2)$.

These properties are initially true because there are implicit Delete operations with time stamps $-\infty$ and $+\infty$, and no attribute update messages have been sent to any federate. Thus, property 2 is maintained, and property 1 does not apply. From Figure 6 it is easy to see that the algorithm described above maintains both of these properties after each new Add and Delete operation. An Update operation with time stamp T also maintains this property by sending a message to each federate (and no others) subscribed to the attribute at time T.

11.5. Memory Reclamation

In addition to the above operations, a mechanism is required to reclaim memory used by the distribution lists and logs. The current federate time of the owner of the attribute plus that federate's lookahead provides a lower bound on the time stamp of any future **Update Attribute Values** service request. Assuming Add and Delete operations also adhere to lookahead constraints, the current time of any federate that can Add/Delete the attribute (i.e., any federate in the federation) plus its lookahead gives a lower bound on future add and delete requests that will be generated by that federate. Thus, the minimum of these values across all federates (analogous to Global Virtual Time in Time Warp simulations) gives a lower bound on the time stamp of any Add, Delete, or Update operations that will be made on the distribution list in the future. If this minimum time stamp value is T, then all tuples with time stamp less than T can be discarded, provided the RTI retains a copy of the distribution list of the attribute A at time T, i.e., some representation of D(A,T) is required. Value tuples with time stamp less than T may be discarded and their storage reclaimed.

11.6. Simplifications

The mechanism described above requires a log of previously sent messages. This log can be eliminated if certain restrictions are made on federate behavior. Recall that the log was required because Update messages containing a time stamp T may be generated before a subsequent Add or Delete operation containing a time stamp smaller than T. If one makes the restriction that at each instant in the federation execution, a global time stamp value T_{bound} exists such that no update messages with time stamp larger than T_{bound} can be sent, and no Add or Delete operation with time stamp less than T_{bound} can be generated, this situation cannot occur. Realization of this mechanism requires a protocol to advance T_{bound} because all federates must agree to an advance of

T_{bound} before it can take effect. An interface similar to that used for Time Advances (i.e., T_{bound} advance and grant services) could be used for this purpose.

A more modest simplification results if *each* federate is constrained to issue Add and Delete operations in time stamp order, and a communications protocol is used that guarantees messages are delivered in the order that they are sent. Specifically, cases 1 and 4 in Figure 6 can be replaced by "null operations" because a federate (un)subscribing to an attribute to which it is already (un)subscribed has no affect. This simplification eliminates accesses to modify the distribution list for cases 1 and 4. A drawback of this simplification is that it places a severe restriction on optimistic federates that do not normally generate operations in time stamp order, due to rollbacks.

11.7. Optimistic Time Management Services

The mechanisms described above supports inclusion of optimistic federates. Optimistic federates must be able to undo operations when a rollback occurs. The HLA retraction mechanism already provides a mechanism to undo **Update Attribute Values** and **Send Interaction** service requests. The Add operation discussed above can be undone by a Delete operation with the same time stamp, and a Delete operation can be undone with an Add operation since composition of these two operations (with the same parameters) is a null operation.

12. HLA-TM Architecture

The time management architecture of the HLA can be viewed according to the logical structure depicted in Figure 7.

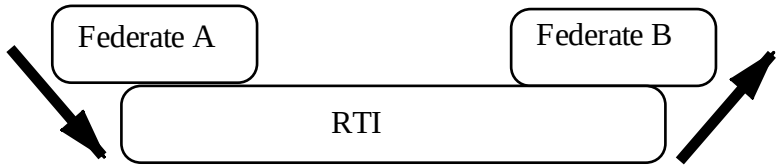

Figure 7. Logical flow of messages using the RTI.

The movement of messages among the federates and the RTI is determined by attributes of the messages that pass through the components and the state of variables that the RTI maintains for each federate. The functioning of the components is illustrated below by example; similar messages are inserted into the architecture in the face of different variable states. In all cases, the situation illustrated is where Federate A sends information (message) for which Federate B is eligible to receive.

The RTI maintains several state variables for each Federate:

- **Time Regulating**: Whether the federate proposes to contribute to the determination of the logical time of other federates.
- **Time Constrained**: Whether the federate accepts input from other federates in determining its own logical time.
- **Local Time**: The time of a federate.
- **LBTS**: Lower bound on the time stamp of any future time stamp ordered message
- **FIFO Queue**: A First-in, First-out queue of messages waiting to be transmitted to a federate.

- **TSO Queue**: A Time Stamp Order queue of messages waiting to be transmitted to a federate.

When sending messages to the RTI, the federate assigns two attributes to the transmission
- **Ordering Service**: The ordering of the message requested by the originating federate. For purposes of these examples, the options are Time Stamp Order (TSO) and Receive Order (RO)
- **Timestamp**: The time assigned to the message by the originating federate.

12.1. Logical Time Federation

Here, both federates are constrained by the time of other federates (Time Constrained) and both federates contribute to the time evolution of other federates (Time Regulating).

State variable	**Federate A**	**Federate B**
Time Regulating	True	True
Time Constrained	True	True
Local Time	10	10
LBTS	12	12
FIFO Queue	empty	msg1, msg2
TSO Queue	empty	msg3@10, msg4@13

Action	**Explanation**
• Federate A sends a TSO message with time stamp 9 • RTI rejects the message	Time stamps for TSO messages must not be less than local time.
• Federate A sends an RO message with any time stamp (msg5) • RTI places it on the FIFO queue for Federate B after msg2 • Federate B does Time Advance Request to 12 RTI passes messages to Federate B as msg1, msg2, msg5, msg3	RTI ignores the time stamp for RO messages
• Federate A sends a TSO message with time stamp 11 (msg5@11) • RTI validates the time stamp and places it on the TSO queue for Federate B between msg3 and msg4 • Federate B does Time Advance Request to 12 • RTI passes messages to Federate B as msg1, msg2, msg3, msg5	

12.2. Logical Time to Unconstrained Federation

In this federation, Federate A (the originator) is Time Constrained and Time Regulating, while Federate B (the receiver) is neither.

State variable	**Federate A**	**Federate B**
Time Regulating	True	False
Time Constrained	True	False

Local Time	10	10
LBTS	12	12
FIFO Queue	empty	msg1, msg2
TSO Queue	empty	empty

Action	**Explanation**
• Federate A sends a TSO message with time stamp 9 • RTI rejects the message	Time stamps for TSO messages must not be less than local time.
• Federate A sends an RO message with any time stamp (msg5) • RTI ignores the time stamp and places it on the FIFO queue for Federate B after msg2 • RTI passes messages to Federate B as msg1, msg2, msg5	RTI ignores the time stamp for RO messages
• Federate A sends a TSO message with time stamp 11 (msg5@11) • RTI validates the time stamp and places it on the FIFO queue for Federate B after msg2 • RTI passes messages to Federate B as msg1, msg2, msg5	RTI treats all messages as FIFO for receiving Federates whose Time Constrained state is False.

12.3. Unconstrained to Logical Time Federation

In this federation, Federate B (the receiver) is Time Constrained and Time Regulating, while Federate A (the sender) is neither.

State variable	**Federate A**	**Federate B**
Time Regulating	False	True
Time Constrained	False	True
Local Time	10	10
LBTS	12	12
FIFO Queue	empty	msg1, msg2
TSO Queue	empty	msg3@10, msg4@13

Action	**Explanation**
• Federate A sends a TSO message with time stamp 9 • RTI rejects the message	Time stamps for TSO messages must not be less than local time.
• Federate A sends an RO message with any time stamp (msg5) • RTI ignores the time stamp and places it on the FIFO queue for Federate B after msg2 • RTI passes messages to Federate B as msg1, msg2, msg5	RTI ignores the time stamp for RO messages
• Federate A sends a TSO message with time stamp 11 (msg5@11) • RTI ignores the time stamp and places it on the FIFO queue	RTI ignores time stamps and treats all messages as FIFO for originating

for Federate B after msg2 • Federate B does Time Advance Request to 12 • RTI passes messages to Federate B as msg1, msg2, msg5, msg3	Federates whose Time Regulating state is False.

12.4. Unconstrained Federation

In this federation, neither federate is Time Constrained or Time Regulating.

State variable	**Federate A**	**Federate B**
Time Regulating	False	False
Time Constrained	False	False
Local Time	10	10
LBTS	12	12
FIFO Queue	empty	msg1, msg2
TSO Queue	empty	empty

Action	**Explanation**
• Federate A sends an TSO message with any time stamp (msg5) • RTI ignores the time stamp and places it on the FIFO queue for Federate B after msg2 • RTI passes messages to Federate B as msg1, msg2, msg5	RTI ignores time stamps and treats all messages as FIFO for originating Federates whose Time Regulating state is False.
• Federate A sends a RO message with any time stamp • RTI ignores the time stamp and places it on the FIFO queue for Federate B after msg2 • RTI passes messages to Federate B as msg1, msg2, msg5	Federates whose Time Constrained state is false are not required to advance time in order to obtain messages.

13. Baseline Definition

The baseline definition of the HLA time management structure includes:

1. *message order:* receive order and time stamp order are included. Priority, causal, and causal and totally ordered are not included. Order is specified by the message sender.
2. *time advance mechanisms:* all of the time advance mechanisms (**Next Event Request**, **Time Advance Request**, and **Time Advance Grant)** are included in the baseline definition.
3. *declaration management:* the mechanism to ensure properly logical time synchronized subscriptions and unsubscriptions of federates is not included in the baseline definition. Proper synchronization of declaration management mechanisms for time stamp ordered messages must be performed by federates.
4. *optimistic time management services:* the **Flush Queue Request** and **Flush Queue Grant** optimistic time management services are included in the baseline definition.

14. Future Revisions

This document contains the official definition of HLA time management services and supersedes all other publications on this subject. Modifications to these services and mechanisms are anticipated as new requirements become apparent. To be included in the HLA, such modifications must:

1. provide significant new functionality that cannot be reasonably realized with existing services, or enable significant performance enhancements to time management or other HLA mechanisms,
2. have a reasonably efficient implementation approach defined,
3. be applicable to a reasonably broad class of actual or envisioned simulations, and
4. have application to specific DoD simulation(s) either in existence or under development.

Future revisions to the time management services and mechanisms will be taken under advisement by the time management working group which will produce a recommendation concerning its inclusion.

15. References

[Birm91] K. Birman, A. Schiper and P. Stephenson, Lightweight Causal and Atomic Group Multicast, *ACM Transactions on Computer Systems,* 9(3): 272-314, August 1991.

[Chan79] K. M. Chandy and J. Misra, Distributed Simulation: A Case Study in Design and Verification of Distributed Programs, *IEEE Transactions on Software Engineering*, SE-5(5): 440-452, September 1979.

[DIS94] The DIS Vision, A Map to the Future of Distributed Simulation, Institute for Simulation & Training, Orlando FL, May 1994.

[Fuji90] R. M. Fujimoto, Parallel Discrete Event Simulation, *Communications of the ACM*, 33(10): 30-53, October 1990.

[Jeff85] D. R. Jefferson, Virtual Time, *ACM Transactions on Programming Languages and Systems*, 7(3): 404-425, July 1985.

[Lamp78] L. Lamport, Time, Clocks, and the Ordering of Events in a Distributed System, *Communications of the ACM*, 21(7): 558-565, July 1978.

16. Glossary of Terms

Note: Also see Section 3 for other definitions.

cancellation: a mechanism used in optimistic synchronization mechanisms such as Time Warp [Jeff85] to delete a previously scheduled event. Cancellation is a mechanism used within the Time Warp executive, and is normally not visible to the federate. It is implemented (in part) using the RTI's event retraction mechanism.

CATOCS: causal and totally ordered; a partial ordering that is causally ordered (see definition of causal order) and also ensures that all receives of a set of messages receive those messages in the same order.

causal order: a partial ordering of messages based on the "causally happens before" ($\rightarrow$) relationship [Lamp78]. A message delivery service is said to be causally ordered if for any two messages M_1 and M_2 (containing notifications of events E_1 and E_2, respectively) that are delivered to a single federate where $E_1 \rightarrow E_2$, then M_1 is delivered to the federate before M_2.

constrained simulation: a simulation where time advances are paced to have a specific relationship to wallclock time. These are commonly referred to as real-time or scaled-real-

time simulations. Here, the terms *constrained simulation* and *(scaled) real-time simulation* are used synonymously. Human-in-the-loop (e.g., training exercises) and hardware-in-the-loop (e.g., test and evaluation simulations) are examples of constrained simulations.

coordinated time advancement: a time advancement mechanism where logical clock advances within each federate only occur after some coordination is performed among the federates participating in the execution, e.g., to ensure that the federate never receives an event notice in its past. ALSP, for example, uses coordinated time advancement.

conservative synchronization: a mechanism that prevents a federate from processing messages out of time stamp order. This is in contrast to *optimistic* synchronization. The Chandy/Misra/Bryant null message protocol [Chan79] is an example of a conservative synchronization mechanism.

event notice: a message containing event information.

Greenwich Mean Time (GMT): Mean solar time for the Greenwich meridian, counted from midnight through 24 hours. Also called "Universal Time [Coordinated]" (UTC).

happens before, causal ($\rightarrow$): a relationship between two actions A_1 and A_2 (where an action can be an event, an RTI message send, or an RTI message receive) defined as follows: (i) if A_1 and A_2 occur in the same federate/RTI, and A_1 precedes A_2 in that federate/RTI, then $A_1 \rightarrow A_2$, (ii) if A_1 is a message send action and A_2 is a receive action for the same message, then $A_1 \rightarrow A_2$, and (iii) if $A_1 \rightarrow A_2$ and $A_2 \rightarrow A_3$, then $A_1 \rightarrow A_3$ (transitivity).

happens before, temporal ($\rightarrow_t$): a relationship between two events E_1 and E_2 defined as follows: if E_1 has a smaller time stamp than E_2, then $E_1 \rightarrow_t E_2$. The RTI provides an internal tie-breaking mechanism to ensure (in effect) that no two events observed by a single federate contain the same time stamp.

independent time advancement: a means of advancing federate time where advances occur without explicit coordination among federates. DIS is an example of a federation using independent time advancement.

LBTS: lower bound on the time stamp of the next time stamp ordered (TSO) message to be received by an RTI from another federate. Messages with time stamp less than LBTS are eligible for delivery by the RTI to the federate without compromising time stamp order delivery guarantees. TSO messages with time stamp greater than LBTS are not yet eligible for delivery. LBTS is maintained within the RTI using a conservative synchronization protocol.

local time: the mean solar time for the meridian of the observer.

lookahead: a value used to determine the smallest time stamped message using the time stamp ordered service that a federate may generate in the future. If a federate's current time (i.e., federate time) is T, and its lookahead is L, any message generated by the federate must have a time stamp of at least T+L. In general, lookahead may be associated with an entire federate (as in the example just described), or at a finer level of detail, e.g., from one federate to another, or for a specific attribute. Any federate using the time stamp ordered message delivery service must specify a lookahead value.

Mean Solar Time: a time measurement where time is measured by the diurnal motion of a fictitious body (called "mean Sun") which is supposed to move uniformly in the celestial Equator, completing the circuit in one tropical year. Often termed simply "mean time". The mean Sun may be considered as moving in the celestial Equator and having a right ascension equal to the mean celestial longitude of the true Sun. At any given instant, mean solar time is the hour angle of the mean Sun. In civil life, mean solar time is counted from the two

branches of the meridian through 12 hours; the hours from the lower branch are marked a.m. (ante meridian), and those from the upper branch, p.m. (post meridian). In astronomical work, mean solar time is counted from the lower branch of the meridian through 24 hours. Naming the meridian of reference is essential to the complete identification of time. The Greenwich meridian is the reference for a worldwide standard of mean solar time called "Greenwich Mean Time" (GMT) or "Universal Time [Coordinated]" (UTC).

message: a data unit transmitted between federates containing at most one event. Here, a message typically contains information concerning an event, and is used to notify another federate that the event has occurred. When containing such event information, the message's time stamp is defined as the time stamp of the event to which it corresponds. Here, a "message" corresponds to a single event, however the physical transport media may include several such messages in a single "physical message" that is transmitted through the network.

message (event) delivery: invocation of the corresponding service (**Reflect Attribute Values, Receive Interaction, Instantiate Discovered Object,** or **Remove Object**) by the RTI to notify a federate of the occurrence of an event.

optimistic synchronization: a mechanism that uses a recovery mechanism to erase the effects of out-of-order event processing. This is in contrast to *conservative* synchronization. The Time Warp protocol [Jeff85] is an example of an optimistic synchronization mechanism. Messages sent by an optimistic federate that could later be canceled are referred to as optimistic messages.

retraction: an action performed by a federate to unschedule a previously scheduled event. Event retraction is visible to the federate retracting the message. Unlike "cancellation" that is only relevant to optimistic federates such as Time Warp, "retraction" is a facility provided to the federate. Retraction is widely used in classical event oriented discrete event simulations to model behaviors such as preemption and interrupts.

real time: The actual time in which a physical process occurs.

real-time simulation: same as constrained simulation.

scaled wallclock time: a quantity derived from wallclock time defined as offset + [rate * (wallclock time - time of last exercise start or restart)]. All scaled wallclock time values represent points on the federation time axis. If the "rate" factor is k, scaled wallclock time advances at a rate that is k time faster than wallclock time.

time: The measurable aspect of duration. Time makes use of scales based upon the occurrence of periodic events. These are: the day, depending on the rotation of the Earth; the month, depending on the revolution of the Moon around the Earth; and the year, depending upon the revolution of the Earth around the Sun. Time is expressed as a length on a duration scale measured from an index on that scale. For example: 4p.m. local mean solar time means that 4 mean solar hours have elapsed since the mean Sun was on the meridian of the observer.

time flow mechanism: the approach used locally by an individual federate to perform time advancement. Commonly used time flow mechanisms include event driven (or event stepped), time driven, and independent time advance (real-time synchronization) mechanisms.

time stamp order (TSO): a total ordering of messages based on the "temporally happens before" ($\rightarrow_t$) relationship. A message delivery service is said to be time stamp ordered if for any two messages M_1 and M_2 (containing notifications of events E_1 and E_2, respectively) that are delivered to a single federate where $E_1 \rightarrow_t E_2$, then M_1 is delivered to the federate before M_2. The RTI ensures that any two TSO messages will be delivered to all federates receiving

both messages in the same relative order. To ensure this, the RTI uses a consistent tie-breaking mechanism to ensure that all federates perceive the same ordering of events containing the same time stamp. Further, the tie-breaking mechanism is deterministic, meaning repeated executions of the federation will yield the same relative ordering of these events if the same initial conditions and inputs are used, and all messages are transmitted using time stamp ordering.

transportation service: an RTI provided service for transmitting messages between federates. Different categories of service are defined with different characteristics regarding reliability of delivery and message ordering.

true global time: A federation-standard representation of time synchronized to GMT or UTC (as defined in this glossary) with or without some offset (positive or negative) applied.

unconstrained simulation: a simulation where there is no explicit relationship between wallclock time and the rate of time advancements. These are sometimes called "as-fast-as-possible" simulations, and these two terms are used synonymously here. Analytic simulation models and many constructive "war game" simulations are often unconstrained simulations.

Universal Time [Coordinated] (UTC): The same as Greenwich Mean Time. A nonuniform time based on the rotation of the Earth, which is not constant. Usually spoken as, "Coordinated Universal Time".