DESIGN AND IMPLEMENTATION OF HLA TIME MANAGEMENT IN THE RTI VERSION F.0

Christopher D. Carothers
Richard M. Fujimoto

College of Computing
Georgia Institute of Technology
Atlanta, Georgia 30332-0280

Richard M. Weatherly
Annette L. Wilson

The MITRE Corporation
7525 Colshire Drive
McLean, Virginia 22102-3481

ABSTRACT

The DoD High Level architecture (HLA) has recently become the required method for the interconnection of all DoD computer simulations. The HLA addresses the rules by which simulations are designed to facilitate interoperability, the method by which information exchanged between simulations is described, and a standard set of software services provided by a common Runtime Infrastructure (RTI). The RTI is responsible for the coordination of collections of cooperating simulations. The familiarization version of the RTI, dubbed F.0., was eveloped at the MITRE Corporation. The purpose of this version is to (i) educate the user community in the correct usage of the RTI, (ii) serve as a platform for introducing new ideas and concepts that will improve quality and effectiveness of the RTI, and (iii) provide a capability until commercial RTIs are available. One of the core components of the RTI is Time Management and is the focus of this paper. In particular, we present the design and algorithms used to implement the HLA Time Management Services in F.0.

1 INTRODUCTION

To address the need for interoperability between new and existing U.S. Department of Defense simulations, the Defense Modeling and Simulation Office (DMSO) commissioned the High Level Architecture initiative. This effort, which began in March 1995, has culminated in the approval by the Executive Council for Modeling and Simulation (EXCIMS) on September 5, 1996. On that same day the EXCIMS also unanimously approved a schedule and procedure for ensuring HLA compliance of all DoD simulations. This HLA baseline definition was then forwarded to the Under Secretary of Defense for Acquisition and Technology [USD(A&T)] for approval and appropriate policy action. On September 10, 1996 the USD(A&T) designated the HLA as the standard technical architecture for all DoD simulations.

The HLA consists of 3 parts: (1) HLA rules, (2) HLA Interface Specification (IFSpec), and (3) Object Model Template. The software that meets the interface requirements set forth by the IFSpec is called the Runtime Infrastructure (RTI). For more information on the HLA and its parts we refer the reader to the DMSO web-page at `http://www.dmso.mil`.

The familiarization version of the RTI, dubbed *F.0*, was developed at the MITRE Corporation. The purpose of this version is to (i) educate the user community in the correct usage of the RTI, (ii) serve as a platform for introducing new ideas and concepts that will improve quality and effectiveness of the RTI, and (iii) provide a capability until commercial RTIs are available.

Within the RTI, one of the core components concerns how time is managed and is the focus of this paper. Here, we describe the design decisions and algorithmic implementation of RTI F.0 that support the HLA Time Management Services. Section 2 sets the stage for this discussion by presenting the design requirements for Time Management and the key properties used to differentiate federate types. Based on these requirements, the RTI's time management services are described in Section 3 and their corresponding algorithmic implementation is described in Section 4. Section 5 concludes this study and presents directions for future work in Time Management.

2 REQUIREMENTS

To achieve the HLA's interoperability requirements, time management *must* appear transparent to all federates engaged in a given federation. This transparency is needed to facilitate federate integration with a federation as well as keep federate developers from having to understand the intricacies of distributed systems programming, thus reducing development efforts and costs. In supporting this transparency, the RTI must allow for: (i) federates that may or may not be constrained by a time-flow mechanism, (ii) constrained federates with different and dynamically changing time-flow mechanisms, e.g. event driven or time-stepped federates or those that switch between these two during execution,

Time Constrained \ Time Regulating	TRUE	FALSE
TRUE	Strict Time Synchronized Federate	Monitor or Federation Management Tool
FALSE	Aggressive Time Synchronized Federate	Externally Synchronized Federate

Figure 1: Simulator Types Based on the *time constrained* and *time regulating* Properties

(iii) federates executing on parallel/distributed platforms using a conservative or optimistic synchronization protocol including those that dynamically change between them, (iv) federates with different event ordering requirements, e.g, receive or time stamp order, (v) federates using a mixture of event orderings and transportation services (e.g. reliable or best effort) and, (vi) dynamically changing federations where federates can join and resign from a federation at any time.

Motivated by the above requirements list, we divided the range of federate types into four major categories based on two key properties that are used in the *Aggregate Level Simulation Protocol (ALSP)* (Weatherly et. al. 1991): (i) *time constrained* and (ii) *time regulating* (see Figure 1). Federates that are *time constrained*, must receive events in time stamp order. Non-time constrained federates receives events in the order that they were received by the RTI. Federates that are *time regulating*, schedule events that must be received in time stamp order. *Non-time regulating* federates schedule events that will be delivered in the order received by the destination. Note, the order of event delivery need not be the order in which events are processed. Federates can reorder events once delivered.

The first major federate category is *strict time synchronized.* To be in this category, a federate must be *time constrained* and *time regulating.* Example federates include simulations synchronized by conservative (e.g. ALSP) and/or optimistic (e.g. Time Warp (Jefferson 1985)) protocols. Optimistic federates are included here because despite allowing out-of-time stamp order execution, the order of committed events (i.e., events not rolled back) strictly adheres to the time stamp order processing rule.

The next major federate category is *aggressive time synchronized.* Here, federates are not *time constrained*, but still schedule time stamped events. A federate falling into this category would be a DIS federate (DIS Steering Committee, 1994) that would like to interoperate with other ALSP federates. In this example, the DIS federate is able to process events in "real-time" and not have its time advance constrained by the progress of other federates in the federation. The caveat is that the DIS federate could receive events in its past and process events out of order, which is allowed by the DIS protocol. Because, ALSP requires strict time stamp order processing of events, the DIS events sent to ALSP federates must be time stamped. By having a *time regulating*, DIS federate, events will be properly time stamped.

The third major federate category is *monitor.* This kind of federate is *time constrained*, but only schedules receive order events to be sent to other federates. The reasoning for the viewer category is that federations will need management tools that display time stamped information, but would schedule "out-of-time" receive order events, such as forcing a federate to resign from the federation or requesting certain status information, such as the length of the RTI's event queues.

The last major federate category is *externally synchronized.* These federates are synchronized by some mechanism outside of the RTI. Consequently, the are not constrained by the RTI's time management services and schedule only receive order events. Here, the RTI serves as a message passing library and time management services, while invoked, do not manage the federate's advancement of simulated time. This category of federates would be used in a DIS federation, such as Synthetic Theater of War (STOW) (Aronson 1996), which uses a real-time clock that is external to the RTI to control the advance of simulated time. Because this federation contains non-time constrained federates, there is no need to schedule time stamp events, thus all federates have their *time regulating* property turned off.

Additionally, the RTI must not only allow federates of different types to interoperate but also allow federates to change their type by toggling the *time constrained* and *time regulating* properties during federation execution.

3 TIME MANAGEMENT SERVICES

In this section, we review the HLA Time Management Service set forth in the IFSpec. These services encompass two aspects of federation execution: (i) *Transportation services* and (ii) *Time advance services.* Each of these is discussed in the sections below.

3.1 Transportation Services

The categories of transportation service are distinguished by (i) reliability of event delivery, and (ii) event ordering. With respect to reliability, version F.0 of the RTI offers two levels of service.

Reliable delivery guarantees that any message sent using this service will arrive intact at the proper destination. This level of reliability comes at the cost of increased latency. Example federates using this service would be ALSP.

Best effort delivery reduces latency, but does not guarantee delivery. Example federates using this service would be DIS.

Event ordering characteristics specify the order and time at which events may be delivered to federates and are central to the HLA time management services. In accordance with the 1.0 IFSpec, F.0 offers two ordering mechanisms.

Receive order events are passed to the federate in the order that they were received. Logically, incoming events are placed at the end of the a first-in-first-out (FIFO) queue, and are passed to the the federate by removing them from the front of this FIFO queue.

time stamp order (TSO) events utilizing this service will be delivered to *time constrained* federates in time stamp order. Further, the time advance services guarantee that no event is delivered to a federate "in its past", i.e., no TSO event is delivered that contains a time stamp less than the federate's current logical time. A conservative synchronization protocol is used to implement this service. TSO events sent to *non-time constrained* federates will be delivered as receive order events.

Earlier versions of the HLA Time Management Services called for the following additional ordering services: (i) **priority order**, (ii) **causal order** and (iii) **causal and totally order**. After several iterations through the IFSpec approval process, these services were excluded from the HLA Baseline Definition (IFSpec 1.0). Because the F.0 design team's charter was to build an RTI that strictly adheres to this baseline definition these additional services were not be included in F.0.

3.2 Time Advance Services

The time advance services serve several purposes. First, it provides a protocol for the federate and RTI to jointly control the advancement of logical time. The RTI can only advance the *time constrained* federate's logical time to T when it can guarantee that all TSO events with time stamp less than or equal to T have been delivered to the federate. At the same time, conservative federates must delay processing any local event until their logical time has advanced to the time of that event, while optimistic federates will aggressively process events and rollback when it receives a TSO event in its past and use T as an estimate of global virtual time (GVT) for fossil collection.

To insure the RTI properly delivers TSO events, a conservative synchronization protocol implements the TSO event delivery service and is used to advance logical time. The principal task of the protocol is to determine a value called *Lower Bound Time Stamp (LBTS)* for each federate, which is defined as a lower bound on the time stamp of future TSO events that it will receive from other federates.

Among *time constrained* federates, there are three subclasses of federates: (i) *conservative event-driven*, (ii) *conservative time-stepped*, and (iii) *optimistic*. For each of these federate subclasses, the following three time advance service have been devised.

Time Advance Request with parameter t requests an advance of the federate's logical time to t. This service is intended to be used by conservative time-stepped federates where t denotes the time of the next time step. Invocation of this service by a *time constrained* federate implies that the following events are eligible for delivery to the federate: (i) all receive order events, and (ii) all TSO events with the same time stamp that are less than or equal to t. When the RTI can guarantee that it has passed to the federate all such events, the RTI invokes the **Time Advance Grant** service, notifying the federate that its logical time has been advanced to t. Upon receiving the grant, the federate may proceed to the next time step.

Next Event Request with parameter t requests an advance to logical time t, or the time stamp of the next TSO event from the RTI, whichever is smaller. By invoking this request, the federate guarantees it will not produce any new TSO messages in the future with a time stamp less than t plus the federate's lookahead if that federate does not receive any additional TSO messages. This service is intended to be used by conservative event-driven federates where t denotes the time of the next local event within the federate that is to be processed. After invocation of this service, the RTI will deliver all receive order events and either (i) deliver the next TSO event (and all other time stamp order events with exactly the same time stamp) if that event's time stamp is less than or equal to t and advance logical time to the time of that TSO event, or (ii) not deliver any TSO events and advance logical time to t. In either case, the RTI calls the **Time Advance Grant** service, notifying the federate of the completion of this request and that logical time has been advanced.

Flush Queue Request with parameter t requests an advance to logical time t, or the time stamp of the next TSO event, or LBTS, whichever is smaller and deliver all receive order and TSO events currently residing within the RTI. This service is intended to be used by optimistic federates, where t denotes the time of the next local event within the federate that is to be processed. After invoking this service, the RTI will deliver all TSO events regardless of their time stamp, should any exist, then advance logical time to the minimum of t, LBTS, and the smallest time stamp of any delivered TSO event and then call the **Time Advance Grant** service, notifying the federate that logical time has been advanced.

To support optimistic federates, the RTI provides a **Retract** service that allows federates to "unschedule" previously sent events. When a federate schedules an event, the RTI provides the federate with a **Event Retraction Handle**. To retract an event, the federate invokes the **Retract** service with the appropriate **Event Retraction Handle** as the parameter. If the retracted event has already been passed to the federate, the retract service is forwarded to that federate.

For *non-time constrained* federates, the time advance services will deliver all receive order events (recall that for *non-time constrained* federates all events are treated as receive order) and immediately advance the federate's logical time to the requested time via the **Time Advance Grant** service.

A lookahead value is specified for each federate. Lookahead defines the minimum distance into the future that a TSO message will be scheduled. A federate may change its lookahead during the federation execution, however, a decrease in the federate's lookahead by an amount k does not take effect until the federate has advanced k units of time.

An important assumption in using these time advance services is that federates must have non-zero lookahead. Because each service will not advance time until the RTI can unconditionally guarantee that *all* events less than or equal to some time t have been delivered, federates with a zero lookahead may cause the federation to deadlock (Fujimoto 1996). Changes to allow zero lookahead federates have been developed and should appear in RTI version 1.0.

4 IMPLEMENTATION

In designing the algorithms necessary to support all the requirements placed on the RTI by an unpredictable federate, every effort was made to avoid RTI modality. The RTI must be able to honor service calls in any order. This leads naturally to a finite state machine design philosophy for the time manager. The completion of every service call will leave the time manager in a position to accept any subsequent service call. The context of service calls, if any, is captured in the state variables of the time manager. As part of this design philosophy, the time management algorithms were constructed to ensure reliability and to facilitate implementation. Consequently, these algorithms avoid the use of potentially unreliable distributed programming constructs, such as global synchronizations.

While the IFSpec is quite detailed in the regard to the data flow into and out of each HLA Service, the design team made two assumptions to ease the development effort required to implement F.0.

(i) All federates are fully connected by point-to-point, reliable, FIFO communication links. The *Adaptive Communication Environment (ACE)* (Schmidt 1993) was chosen to support the communications layer of F.0 because it supported reliable communications between programs started by different users without having to resort to using "remote-shells", which has been shown to have security problems. Because ACE uses the TCP/IP protocol which is reliable, FIFO, point-to-point, it was decided all algorithms should take advantage of this communications property whenever possible.

(ii) All TSO messages are sent reliable and FIFO. The question of designing a synchronization algorithm that allows best-effort, TSO messages remains unresolved. Due to the F.0 tight delivery schedule, the design team did not have time to adequately investigate this issue. Accordingly, it was decided that all TSO messages must sent over a reliable communication link. The FIFO property assumption is added because the F.0's communications layer supported it, allowing for a straight-forward synchronization algorithm to be used. It should be noted that federates can send best-effort, TSO message, however, the synchronization algorithm currently used will not guarantee that messages will be delivered in TSO order.

In this section, we provide a detailed description of F.0's time management implementation. The first major design constraint concerned the issue of concurrency or "threadedness" within the RTI. This issue and it's implications for F.0 are discussed. Next, we present an overview of F.0's object hierarchy and then devote the remainder of this section to the implementation of F.0's time management algorithms.

4.1 RTI Threading

The predecessors to the F.0 RTI are the 0.1 thru 0.33 versions. In this sequence of prototype RTIs, the core components of the RTI were divided into independent threads of computation taking the form of CORBA servers. This partitioning of the code was along the lines of the RTI services provided (i.e. the Time Manager thread serviced all time advance service requests). CORBA was used to implement prototype RTIs and CORBA IDL was used as the original application programmer interface language. The AMG selected CORBA IDL because of its operating system and programming language independence. The RTI implementation team choose CORBA for the support it provided in the rapid development of distributed systems. The CORBA-based

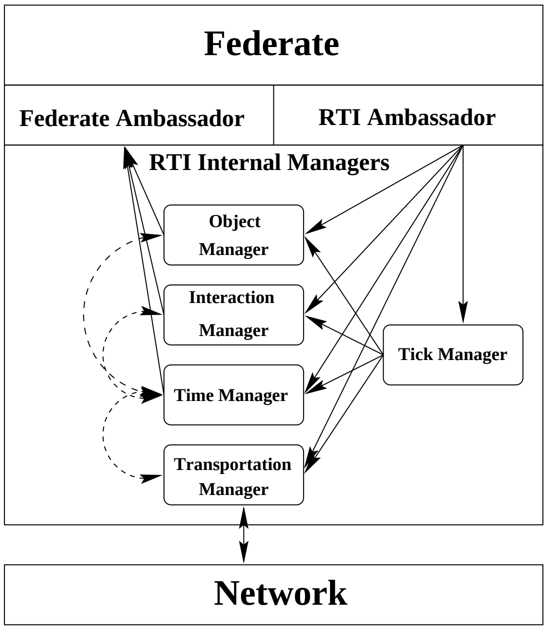

Figure 2: Version F.0's Object Hierarchy

RTI allowed experience to be gained with infrastructure software that was separate from the federate. This was the implementation approach already being taken by many large analytic systems such as JWARS (see `http://afmsrr.sc.ist.ucf.edu/resource/jwars.html`).

When the $0.X$ version series was delivered, it became increasing clear that the performance demands of other systems in the HLA community would not be met, such as ModSAF (see `http://www.ait.nrl.navy.mil/modsaf`). The reason for this lies with the number of costly interprocess communications (IPC) hops taken in the RTI to deliver a single message to the federate.

To address the concerns of performance-demanding federates, the AMG decided that F.0 should be a singly-threaded library that links directly into the federate's program. By having a singly-threaded RTI, the possibility of race conditions and deadlock is reduced. Moreover, time consuming IPCs and context switches are eliminated. However, the consequence of this decision is that it complicates federate implementations. In particular, the federate must explicitly yield control to the RTI so it can process service requests and deliver data to the federate. How this issue is addressed in F.0 is discussed in the next section below.

4.2 Object Hierarchy

The RTI, shown in Figure 2, is composed of three main parts: (i) *RTI Ambassador*, (ii) *Federate Ambassador*, and (iii) *Internal RTI Managers*. The solid lines represent service invocations either from the federate into the RTI or vice versa. The dashed represent intra-RTI service invocations between the different RTI Internal Managers. The first two parts of the RTI are used to pass information from the RTI to the federate and vice versa. The RTI Ambassador serves as an interface for marshalling service requests made by the federate to the appropriate RTI Internal Manager. The Federate Ambassador serves as an interface for marshalling service request responses, such as events or **Time Advance Grants**, from the RTI to the federate. Each federate must provide an implementation of the Federate Ambassador's services. The interface to both the RTI Ambassador and the Federate Ambassador are subject to HLA standards.

The Internal RTI Managers support five categories of run-time services. The *Object Manager* implements the object creation, object destruction, ownership, object publication and object subscription services. The *Interaction Manager* implements the services that create and destroy interactions as well as interaction publication and subscription services. The *Time Manager* implements the services for advancing logical time and will be discussed in detail. The *Transportation Manager* is used to send and receive data and supports the services provided by the other managers. The federation management services are implemented in the RTI Ambassador.

In keeping with the above design constraint, the RTI is designed to be linked in as part of the federate to form a single simulation with only one thread of control. To reduce the effort involved in integrating federates with a single threaded RTI, F.0 provides a **tick** service in the *RTI Ambassador* that gives the thread of control to the RTI and that ensures that all the necessary internal RTI functions and service requests are completed.

When a federate requests the **tick** service, that request is sent to the *Tick Manager*. The *Tick Manager* then invokes the **tick** service provided by each of the RTI Internal Managers to perform their necessary functions. For example, suppose a federate issues a **Time Advance Request** for some time, t. For the RTI to process this request, the **tick** service must be invoked. Upon doing so, the *Tick Manager* would invoke the *Transportation Manager's* **tick** service. Having the single thread of control, the *Transportation Manager* would deliver pending events from the network up to the *Time Manager*, where they are enqueued into the appropriate queue (either TSO or receive order). When the *Transportation Manager's* **tick** service completes, control is returned back to the *Tick Manager* who immediately gives control to the *Time Manager* by invoking its **tick** service. The *Time Manager*, seeing there is a pending **Time Advance Request**, examines the receive order and TSO queues and delivers the appropriate events to either the *Object Manager* or *Interaction Manager*, who then directly forwards the event, if necessary, to the *Federate Ambassador*. Note, the

Tick Manager was bypassed in delivering events from the *Time Manager* to the *Federate Ambassador*. This was done to expedite the event delivery process. After the *Time Manager's* **tick** service completes, the *Tick Manager* gains control and similarly invokes the **tick** service on the *Object* and *Interaction Managers*. Once, the *Tick Manager* has "ticked" all the managers, control is returned to the federate. In practice, the federate will need to invoke the **tick** service more than once (i.e. "poll" the RTI) before the **Time Advance Grant** will be issued.

4.3 Time Management State Transitions

Internal to RTI Time Management there are six states used to determine what actions need to be completed and ensure that these actions occur in the correct order. These six states and their respective transitions are described below:

Initialized: Prior to joining a federation but after the RTI has been initialized, the *Time Manager* is placed in this state.

Joining: When the federate invokes the **Joining Federation** service, the *Time Manager* moves from the *Initialized* state to this state.

Idle: Once the federate joining process is completed, the *Time Manager* transitions to this state.

Time Pending: When the federate invokes the **Time Advance Request** service, the *Time Manager* moves from the *Idle* state to this state. Next, the *Time Manager's* **tick** service (called via the *Tick Manager*) will in turn invoke the **Do Time Pending** decision service (discussed below) based on being in this state and deliver the appropriate receive order and TSO events. Having completed the service request, the **Do Time Pending** service will issue the **Time Advance Grant** and transition back to the *Idle* state.

Event Pending: When the federate invokes the **Next Event Request** service, the *Time Manager* moves from the *Idle* state to this state. Next, the *Time Manager's* **tick** service will in turn invoke the **Do Event Pending** service (see below) based on being in this state and deliver the appropriate receive order and TSO events. Having completed the service request, the **Do Event Pending** service will issue the **Time Advance Grant** and transition back to the *Idle* state.

Flush Pending: When the federate invokes the **Flush Queue Request** service, the *Time Manager* moves from the *Idle* state to this state. Next, the *Time Manager's* **tick** service will in turn invoke the **Do Flush Queue Pending** decision service (see below) based on being in this state and deliver the appropriate receive order and TSO events. Having completed the service request, the **Do Flush Queue Pending** service will issue the **Time Advance Grant** and transition back to the *Idle* state.

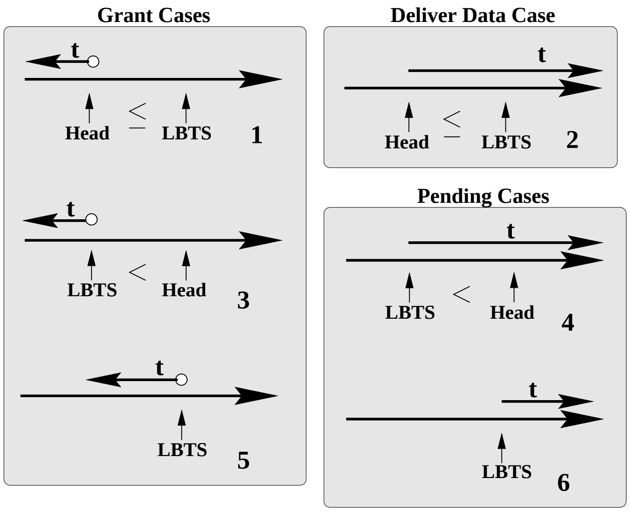

Figure 3: Time Stamp Order Event Queue Case Groupings

4.4 TSO Event Queue Case Groupings

During a time advance request-grant cycle, the RTI must determine which TSO messages are eligible for delivery to the federate. In making this determination, the RTI must consider the relationship between (i) the time stamp of the event at the head of the TSO queue, (ii) the requested time to which the federate wishes to advance and (iii) LBTS. Shown in Figure 3 are the six relationship cases between these three factors, and how a decision is reached as to whether the RTI should (a) grant, (b) deliver a TSO event, or (c) do nothing. **Head** denotes the time stamp of the event at the head of the TSO queue. **LBTS** denotes the lower bound time stamp on any event that could be sent to this federate in the future. **t** denotes the time to which the federated has requested to advance. The white dot is used to denote a strictly less than relationship.

We argue that these six cases provide a complete coverage of the all possible values for these three factors. To make this argument consider the following case groups: (i) cases 1 and 2, (ii) cases 3, and 4, and (iii) cases 5, and 6. We observe that in each of these groups, the value of t, which denotes the request time, spans the entire time line. Thus, each of these groups considers all possible values of t. Next, we observe that groups (i) and (ii) together cover all possible relationships between the the head of the TSO queue, and LBTS. Last, we observe that group (iii) covers the all cases when the TSO queue is empty and no head exists. Consequently, these six cases provide complete coverage.

Now, the case grouping names shown in Figure 3 denote the action generally taking by the *Time Manager* when in that particular case. Consider the **Grant Cases**(cases 1, 3, and 5): we observe that the request

```
Time Advance Request( FederationTime t)
    case state of
        IDLE
            if t < LT then
                throw FederationTimeAlreadyPassed()
            end if
            currentLookahead := max( specifiedLookahead,
                LT + currentLookahead − t)
            requestTime := t
            LT := t
            if timeRegulating then
                send newLT((LT + currentLookahead),
                    requestTime)
            end if
            state := TIME_PENDING
            more := TRUE
            fifoRemainingToDequeue := size of FIFO queue

        EVENT_PENDING
        TIME_PENDING
        FLUSH_PENDING
            error TimeAdvanceAlreadyInProgress
    end case
```

Figure 4: **Time Advance Request** Service

time is always smaller than either the head of the TSO queue or LBTS. Because of this, we know it is safe for the *Time Manager* to issue a **Time Advance Grant** to the request time, t. Likewise, in the **Deliver Data Case** (case 2), since the head of the TSO queue is the smallest of the three factors, we can always deliver the event that is at the head of the TSO queue. Last, in the **Pending Cases** (cases 4 and 6), the RTI can neither grant nor deliver data since both the request time and head of the TSO queue are greater than or equal to LBTS. Note, there are some exceptions to the actions taken by the *Time Manager*, depending on the time advance service.

In the next section, we present the algorithm for servicing a **Time Advance Request**. Due to space limitations, we are unable to present the algorithms used to service **Next Event Request** and **Flush Queue Request**.

4.5 Time Advance Request

When the **Time Advance Request** service is invoked, shown in Figure 4, the *state* of the *Time Manager* is first checked. If the *state* is anything but *Idle* an exception is thrown indicating that a time advance service request is already in progress. Next, because the federate can shrink its lookahead during federation execution, the *Time Manager* must do it in a way that no causality errors result. This gradual shrinking of the lookahead (if necessary) is accomplished by the following expression: $currentLookahead = max(specifiedLookahead, LT + currentLookahead - t)$, where $currentLookahead$ is the actual lookahead used by the *Time Manager*, $specifiedLookahead$ is the smaller lookahead value that has been recently reset by the federate and t is the request time. This expression guarantees that the federate will advance k units of time before shrinking the lookahead by k units.

Then, the request time, t, is stored, and the federate's logical time, LT, is updated. This update of LT may seem inappropriate at this point given that other messages may exist in the TSO queue with a time stamp less than the request time. However, the semantics of this service as stated in the IFSpec stipulate that the federate will be granted to the request time. Moreover, the federate in making this request is stating that any future messages sent to other federates will have a time stamp greater than or equal to $t + currentLookahead$. Consequently, from the IFSpec's point-of-view this behavior is correct, however federates may have events delivered with a time stamp less than LT. Accordingly, we believe the AMG should consider amending the IFSpec to mitigate this ambiguity.

Next, a $newLT$ message that contains $LT + currentLookahead$ and the request time is broadcast to all other federates along the reliable, FIFO, point-to-point communication links. This $newLT$ message can be viewed as a null message from Chandy/Misra's null message algorithm (Chandy et. al. 1979). When another time constrained federate's *Time Manager* receives the $newLT$ message, the logical time plus lookahead value is stored and used to compute LBTS. LBTS for a federate is defined as minimum of all other federates logical clock plus lookahead values and is computed every time a new "LT" value arrives. The $newLT$ message's request time is stored separately and used to break situations where the next event in the federation is far into the future and the federation is creeping ahead by exchanging $newLT$ messages because the lookahead is small. This creeping effect only occurs in the **Next Event Request** and **Flush Queue Request** services.

Having sent the $newLT$ message, the *state* of the *Time Manager* is set to *Time Pending*. Now, to tell the *Tick Manager* that the *Time Manager* has more work to complete, *more* is set to TRUE. Last, the size of the receive order event queue is stored prior to returning control to the federate. The reason for storing the receive order event queue size will discussed below.

When the federates yields control to the RTI via the **tick** service, the *Tick Manager* will invoke the *Time Manager's* **tick** service. In this service, the delivery of all receive order events occurs before any TSO events are considered for delivery. Delivering all the receive order events will take several invocations of this service since only one event is delivered per invocation. Also, as a means of flow-control, this service will only deliver the number of receive order events that is equal to the receive order queue size at the time the request was made. This was done to ensure the amount of time spent processing this service is consistent between successive invocations and to prevent continuous FIFO

```
Do Time Pending()
    case findCases() of
        DELIVER_DATA:
            dequeue one TSO message
            more := TRUE

        GRANT:
            grant LT
            state := IDLE
            more := FALSE

        PENDING:
            fifoRemainingToDequeue := size of FIFO queue
            if fifoRemainingToDequeue = 0 then
                more := FALSE
            else
                more := TRUE
            end if
    end case
```

Figure 5: **Do Time Pending** Decision Service

message arrival from precluding TSO message delivery.

Next, after the receive order events have been delivered, TSO events may be delivered provided the federate is *time constrained*. Because the *state* of the *Time Manager* is *Time Pending*, the **Do Time Pending** decision service (see Figure 5) is invoked, which determines which, if any, TSO events are to be delivered to the federate and then the **Time Advance Grant** is issued. By comparing the request time, the time stamp of the head event of the TSO event queue and LBTS, the proper case grouping can be determined. If the case is *Deliver Data*, this service will deliver a single TSO event and set *more* equal to TRUE. If the case is *Grant*, then a **Time Advance Grant** is issued for the logical time of the federate, LT, the *state* is set to *Idle* and *more* is set to FALSE, indicating this service request is complete and the *Time Manager* has no pending work to perform. If the case is *Pending*, no actions can be taken and the number of receive order events is recalculated to be used in delivering of receive order events in future invocations of the *Time Manager's* **tick** service. If there are receive order events to deliver, *more* is set to TRUE. Otherwise it is set to FALSE.

5 CONCLUSIONS AND FUTURE WORK

In this paper, we present the design and algorithms used to implement the HLA Time Management Services in version F.0 of the RTI. F.0 will be followed by RTI 1.0 in May 1997 and RTI 1.1 in the Fall of 1997. These future releases will address candidate functionality for inclusion in the HLA and services not provided in F.0. In future RTI versions, we hope to address the following such as, (i) zero lookahead federates, (ii) repeatability of results in the presence of zero lookahead federates, (iii) casually correct toggling of *time regulating* properties, and (iv) development of an efficient synchronization protocol that allows best-effort, TSO messages to be sent.

REFERENCES

Aronson, J. 1996. The STOW-97 System Architecture and Implementation Design. *Technical Report CDRL:A005, STOW Web Page at* `http://www.stow.com`.

Chandy, K. M., J. Misra 1979. Distributed simulation: A case study in design and verification of distributed programs. *IEEE Transactions on Software Engineering* SE-5(5):440–452.

DIS Steering Committee 1994. The DIS Vision, A Map to the Future of Distributed Simulation. *Technical Report IST-SP-94-01, Institute for Simulation and Training.*

Fujimoto, R. 1996. HLA Time Management. *Technical report, DMSO HLA Web Page at* `http://msis.dmso.mil/projects/hla/`.

Jefferson, D. R. 1985. Virtual time. *ACM Transactions on Programming Languages and Systems* 7(3):404–425.

Weatherly, R. D. Sidel, and J. Weissman 1991. Aggregate Level Simulation Protocol. *1991 Sumer Computer Simulation Conference.*

Schmidt, D C. 1993. The Adaptive Communication Environment. 11^{th} *Sun User Group conference.*

AUTHOR BIOGRAPHIES

CHRISTOPHER CAROTHERS is a Research Scientist and Ph.D. candidate in the College of Computing at the Georgia Institute of Technology. He was part of the F.0 development team as a summer intern at the MITRE corporation.

RICHARD FUJIMOTO is a professor in the College of Computing at the Georgia Institute of Technology. In addition to his many other research activities in the field of parallel simulation, he chairs the HLA Time Management Committee.

RICHARD WEATHERLY is a is Chief Engineer for The MITRE Corporation's Information Systems and Technology Division. He wrote the first version of the HLA Interface Specification and lead the RTI 0.X, F.0, and 1.0 software development teams.

ANNETTE WILSON is a Senior Systems Engineer at the MITRE Corporation. She served as the Technical Leader for RTI 0.X and F.0 Time Management and Data Distribution components.