# Middleware for Real-Time Distributed Simulations

Thom McLean, Richard Fujimoto,
and Brad Fitzgibbons
*College Of Computing*
*Georgia Institute of Technology*
*Atlanta, GA 30332-0280*
*{mailto:thom, fujimoto, bradf}@cc.gatech.edu*

## Abstract

*Distributed simulation applications often rely on middleware to provide services to support their execution over distributed computing environments. Such middleware spans many levels, ranging from low-level support for data transmission through object request brokers to higher level, simulation specific functionality such as time management. We discuss design alternatives for realizing such middleware for hard real-time (HRT) distributed simulations such as hardware-in-the-loop applications. An implementation of a real-time runtime infrastructure (RTI) middleware is described, and its performance compared against a non-real-time implementation. The context for this work is the High Level Architecture standard that has been defined by the U.S. Department of Defense.*

## 1. Introduction

The High Level Architecture (HLA) has become the standard technical architecture for modeling and simulation in the U.S. Department of Defense. One component of the HLA, the Interface Specification (IFSpec) [1], defines the set of services that are used by individual simulations to interact with each other. HLA simulations use runtime infrastructure (RTI) middleware software to provide services to support interconnecting simulations as well as to manage the distributed simulation execution. In a previous paper we described the RTI-Kit, a set of related libraries for developing RTIs [2]. Previous work on RTI software focused on efficient execution on tightly coupled machines such as shared memory multiprocessors or workstation clusters using high-speed interconnects. In this paper, we turn our attention to specific problems concerning its use for real-time distributed simulation applications.

Composing federations of autonomous simulators or simulation components has become an accepted paradigm to realize distributed simulation systems, especially for real-time interactive simulations. In this type of heterogeneous system, currently, a simulation application or *federate* will not use time management services, preferring to synchronize to local, or *wall-clock time* (WCT). Such distributed simulations typically use unordered, or *receive order* (RO) in HLA terminology, message delivery thereby avoiding overheads associated with time-management and *time-stamp ordered* (TSO) message delivery. However, this comes at a cost: RO message delivery is generally not repeatable, and can cause causal and temporal anomalies, as discussed in [3]. Except for certain types of hardware-in-the-loop (HWITL) simulation systems executing in a real-time operating system, most real-time simulations use RO delivery.

The HLA IFSpec does not specify timeliness criteria for RTI services, with one exception. The tick() call takes optional parameters to specify the minimum and maximum time that the RTI should spend before returning from tick(). This is a soft requirement, however, and the reference clock for the parameter is not specified. The lack of timeliness requirements, or any additional guidance, is a critical limitation for real-time simulation systems where the amount and predictability of RTI overhead is an important design factor. The inability to specify or guarantee message delay, for example, is one factor that has precluded acceptance of HLA in HWITL simulations and testbeds.

In addition to the obvious difficulties with respect to HWITL integration, the lack of timeliness

criteria in HLA aggravates the problem of simulation analysis. For example, causal relationships between RO events are difficult to detect. Because the messages are delivered unordered and without timestamps or message delay information, no causal relationships can be inferred except for trivially implied causality within a single federate. It is common practice for real-time federations to add an additional time attribute to a message for the purpose of dead-reckoning and detecting causal anomalies. The difficulty in analyzing and validating real-time simulation data is a frequent topic in VV&A forums. This is an important problem when trying to use simulation results to support acquisition decisions or course-of-action analysis.

Recognizing these problems, we describe our experiences in providing real-time functionality to the HLA and realizing this functionality within an RTI. Although the HLA is the context for this work, the approaches that are described later are generally applicable to any real-time distributed simulation using a peer-to-peer architecture. In suggesting real-time features, we are not simply facilitating *federate* execution in real-time, rather, we propose modifications to the RTI to allow more discriminate analysis and management of *federation* execution. Moreover, with a more complete understanding of distributed deadlines for execution, we may be able to develop better ways of specifying requirements for RTI performance for a desired model of real-time execution. Specifically, we are proposing changes that specify real-time deadlines for RTI functions, thereby enabling hard real-time federation execution.

The remainder of the paper is organized as follows. Related work is presented in Section 2. Section 3 illustrates common models for real-time execution of distributed simulations. Section 4 presents various architectural alternatives for realizing real-time distributed simulation services. Section 5 describes a prototype implementation that was developed and presents measurements of its performance. Section 6 speaks to the need for changes to the RTI IFSpec. Section 7 discusses some potential benefits of this approach as well as conclusions and future work.

## 2. Related Work

Historically, much of the motivation for Distributed Interactive Simulation (DIS) was the appeal of linking real-time training simulators. The original DIS protocol made no assumptions about message delay or infrastructure overhead. Messages included a sender timestamp, which the receiver could use for dead-reckoning or reordering messages (although reordering was beyond the scope of the protocol.)

Unordered delivery limited the applicability of the DIS protocol. Causal and temporal anomalies were common, especially in executions across wide area networks. Some early DARPA programs, such as WarBreaker, may have overestimated the analytic value of DIS-based simulation. However, these early programs, and more recent initiatives such as simulation-based acquisition (SBA), have established the US Department of Defense and wider interest in using real-time simulation for credible Research, Development, Test and Evaluation (RDT&E).

One of the unfulfilled promises of HLA is the potential for time-managed and non-time-managed federates to interoperate. The earliest reports on HLA performance, however, noted that the infrastructure overhead (even without time management) was too great to be useful for wide area distributed simulation [16].

To date, most work on HLA RTI software has focused on networked workstations using well-established communication protocols such as UDP and/or TCP. While such implementations are sufficient for large portions of the M&S community, some applications require higher communication performance. Special purpose protocols for shared virtual environments have been suggested to address the deficiencies of TCP and UDP. Brutzman, Zyda and others suggested that a virtual reality transfer protocol (VRTP) could be designed to meet requirements of distributed real-time interactions[13],[15]. By combining such services with a high-performance infrastructure [14], they postulate the plausibility of wide-area collaborative environments.

Another approach to meeting real-time simulation requirements has been drawn from work in real-time object databases[10]. Real-time CORBA[11] architectures have been suggested as a mechanism for ensuring high-performance state sharing[12]. Both the VRTP and CORBA approaches attempt to satisfy the application demands implicitly, within the execution

infrastructure. The approach in this paper is different, in that it assumes that the deadlines for real-time execution must be visible to the application.

Shared memory multiprocessors and cluster computing platforms offer high performance alternatives to the relatively high latencies of other distributed approaches. A few systems have been adapted for use in high performance computing platforms as in [4] and [5]. At least one other research group is actively pursuing incorporation of QoS features into an RTI architecture, as reported in [17].

A few other hard real-time HLA and HLA-like RTI implementations, primarily for HWITL testing, have been presented at simulation workshops. However, these efforts have, for the most part, provided non-real-time HLA interfaces to a real-time environment. Additionally, some commercial products claim hard real-time execution (see http://www.opal-rt.com) by providing a real-time operating system (RTOS) simulation execution environment.

## 3. Models of RT Execution

Typically, distributed simulations use local wall-clock time (WCT) as the reference for real-time, ignoring any variance (skew) between local clocks. Clock skew can be accounted for, if required, as in [6]. We frequently say that a simulation is executing in real-time when it maintains a constant relationship between local simulation time and WCT.

### 3.1. Common Models in Non-Real-Time Environments

We note that in order to ensure that real-time simulations remain synchronized with the WCT, they must execute in one of two manners. A simulation may execute at an explicit time step, in which case it will suspend execution until the expiration of the time step interval. Alternatively, a simulation may request the wall clock time at the end of a time step and, based upon the return value, vary the time step interval accordingly. These methods are explained below.

*Time-stepped* execution is the predominant model for "DIS" style applications. The simulation is executed as a series of discrete intervals of length $\Delta t$. In each interval, the application must 1) retrieve incoming messages, 2) compute the new internal state, and 3) send appropriate messages to other logical processes. At the end of this processing cycle, internal time is incremented by $\Delta t$, and the application suspends execution until that time is reached. Timestamps, if present, are not used to order delivery of the messages. All messages are delivered RO and are assumed to be relevant for NOW (the current wall-clock time).

This method of execution has the benefit of providing regular, periodic state updates. Such a method is appropriate for a visualization routine, where each cycle yields a new frame in the animation.

There are a few drawbacks to this approach, however. In order to ensure that a queue of incoming events does not grow, the application must normally flush the incoming event (message) queue in each cycle. Since the intervals are fixed at $\Delta t$ length, all processing must be completed in $\Delta t$ time. However, variance in the number of incoming messages and other factors in the computation may increase the time spent in each processing cycle. Therefore, $\Delta t$ must be at least as large as the longest total time that can be spent processing. The difference between $\Delta t$ and the actual time spent during a cycle is called the *slack time*. This is shown in Figure 1. The existence of positive slack time in each cycle ensures that the simulation time is able to stay correctly synchronized with wall clock time. If the cycle takes longer than $\Delta t$, then simulation time will lag behind wall-clock time.

The *no-wait*, or *time-sampled* model is designed to post updates as quickly as possible. It may be appropriate when response time of the system to any input is the most important factor in the design. It permits synchronization to WCT without the requirement for slack time. We can find examples of this type of system in human-in-the-loop simulators. For this model, the processing cycle may include the same basic functions as in time-stepped. The difference is how simulation time is advanced. In the no-wait model, the current value of wall-clock time is sampled to determine the new simulation time. Messages are again processed in receive order, and are assumed to be relevant for NOW. Notice that in this model there is no need for slack time, because variance in processing

time is accounted for each time the wall-clock time is sampled.

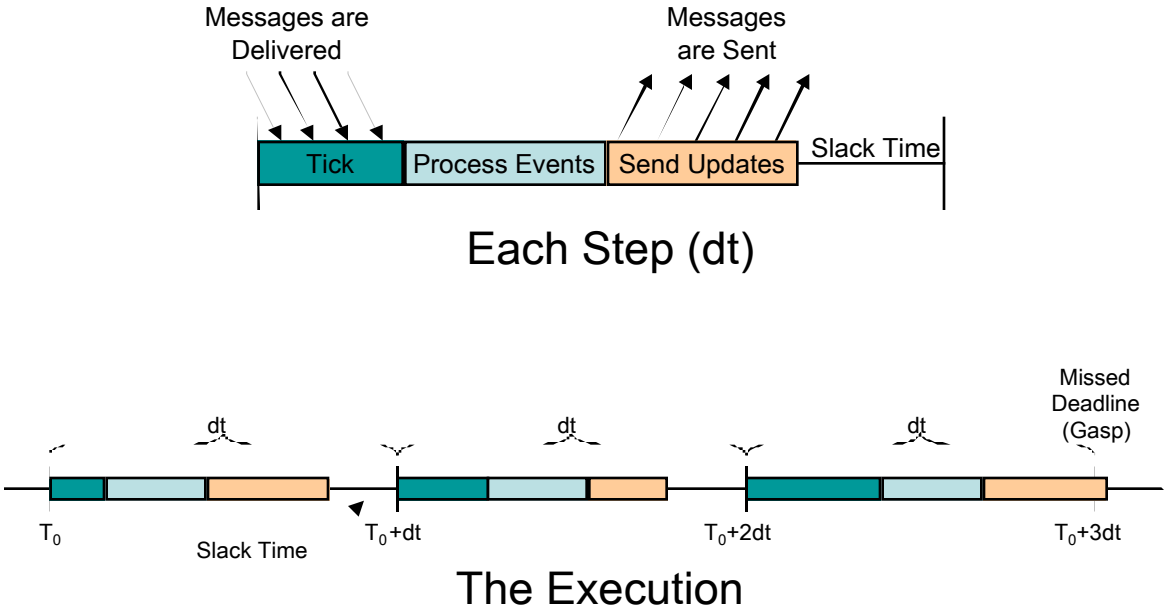


**Figure 1 - Time-Stepped Execution**

### 3.2. Real-Time Scheduled Execution

*Scheduled* execution of processes is available in operating systems with real-time features. Outside the distributed simulation community, distributed real-time computing commonly involves scheduling remote and local events with real-time execution constraints. In this model, all processing is scheduled to be completed by a specified deadline. In a simulation system, a logical process (LP) can schedule processing for itself or for another LP. Messages to remote LPs are simply one class of remotely scheduled processing. Most current simulation systems do not employ such a methodology, but it is this paradigm that offers the most significant change to the state of the art in real-time distributed simulation.

Note that both the time-stepped and time-sampled execution models are easily implemented in a scheduled execution environment. For time-stepped behavior, a process would simply schedule itself for periodic execution. For a time-sampled execution model, the process would re-schedule itself for immediate execution upon completion of the current computation.

Using a scheduled execution model implies that messages must meet delivery deadlines. It also places restrictions on the processing of RTI services. Additionally, we could place real-time requirements on the entire federation execution. Effectively, this would require that the federate and RTI processes be scheduled independently, ensuring federate, RTI, and federation deadlines were all met.

### 3.3. Hard Real-Time Federation Execution

Hard real-time execution implies that meeting a deadline is an absolute requirement for correctness in the simulation system. For a hard real-time *federate*, this constraint could be applied to any one of the execution models above. A hard real-time *federation*, on the other hand, must also meet execution deadlines for federation-level, distributed processes. This new model for federation execution necessitates that RTI processes execute in real-time.

Most real-time simulation systems consider the network or simulation infrastructure to be a system boundary. This design is certainly the simplest way to connect real-time simulators, but many other paradigms are possible. If we extend the real-time model to include the RTI, we can

develop additional temporal constraints for the RTI, permitting greater control over the entire execution. We can easily understand how we might develop service timing bounds similar to those currently available in the tick() call to regulate RTI overhead. We can also easily understand how we might take advantage of bounds on message delivery delay, even in RO processing. But we also may wish to place other real-time constraints on RTI services, or even make the RTI operate asynchronously, acting as a distributed real-time scheduler.

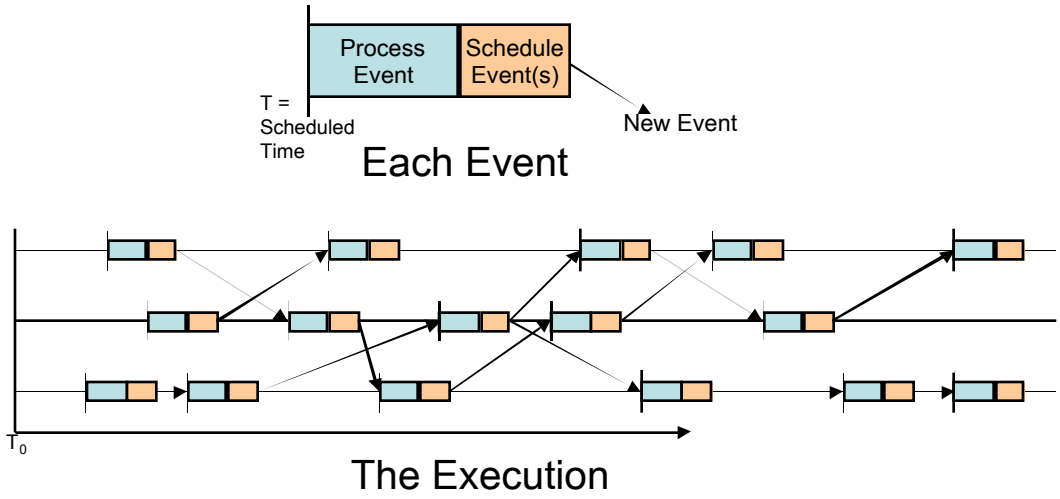


**Figure 2 - Real-Time Scheduled Execution**

To extend our system boundaries and develop hard real-time federation level concepts, we must place some restrictions on the larger execution environment. For this paper, we assume two key conditions. First, we assume a network quality-of-service (QoS) guaranteed latency bound. Secondly, we assume a single ubiquitous WCT, observable to within some bounded skew in any part of the system. We intuitively observe that any notion of distributed deadlines is predicated on both conditions holding.

The execution environment must be able to guarantee an upper bound to communication latency. This requirement is obvious when we consider that, in order to schedule a remote process with a hard real-time deadline, the message that conveys the processing request must be received by some earlier deadline. Thus, the network must provide a system-wide latency guarantee or be capable of negotiating latency QoS between communicating processors. In practice, most commodity networks do not provide QoS. An exception is ATM, which does provide some level of control over latency. However, cluster-computing environments can allow for bounded latencies when utilized well below their bandwidth limits. In fact, as we show in section 6, high-performance communication services, on a simple local area network, can provide very predicable communications, and is suitable for real-time RTI systems.

To meet real-time federation processing deadlines, we assume that processes share a common notion of real-time. This is not an absolute system requirement for real-time execution in that some processors may elect to execute as-fast-as-possible (AFAP). As long as the AFAP federates are capable of executing faster than real-time, a global relationship between federation simulation time and WCT can be maintained, and the federation can be said to execute in real-time. However, for simplicity in this paper, we assume that all processes are executing using some RT model, and therefore are individually managing a relationship between federate simulation time, and (local) WCT. Therefore, in order to provide hard real-time (HRT) deadlines in such a distributed simulation environment, a common notion of WCT is necessary to ensure that deadlines are meaningful throughout the distributed execution.

When both the wall-clock synchronization and latency conditions are met, then a HRT distributed simulation environment may be realized. Using the Georgia Tech RTI-Kit, explained next, we incorporate real-time execution concepts into the RTI design.

## 4. Architectural Alternatives for Real-Time Services

In this section, we explain three general classes of RTI services with respect to real-time processing. The first and simplest class of RT service involves scheduling a process with a remote deadline. For HLA, the canonical example of this service is a real-time guarantee on message delivery. A second, and more complicated RT service class will leverage the communications bounds to provide real-time deadlines to a federate for certain RTI service invocations, even when the services require a distributed computation. A third alternative integrates the real-time services, with a federation execution model, to realize a distributed, hard real-time simulation system within which federate and federation execution deadlines can be specified and managed.

### 4.1. Real-Time Message Delivery Assurance

As a first step to HRT execution, RTIs must begin by providing message delivery assurances. Specifically this means that *a message will be available to be delivered by a specified wall clock time.* This is depicted in Figure 3. Using the current HLA paradigms for TSO and RO messages, this would imply that a federate could receive an interaction or attribute update at least by the next tick() call following the delivery deadline. For TSO messages, the delivery assurances do not immediately obviate the need for conventional simulation time management techniques. However, one might exploit the delivery assurances to develop more efficient time management approaches, as discussed later. Alternatively, by specifying a delivery deadline that is prior to the time stamp, the RTI can guarantee that every message is safe when its timestamp equals WCT.

To implement delivery assurances, the RTI would obtain the quality of service required for the requested delivery performance and would ensure that the message is available to be delivered by the send-time plus the requested bound.

It is important to remember that the delivery assurances do not preclude any desired federate synchronization model. As shown in the figure, message delivery can occur synchronously, during a federate's tick() call, or asynchronously, using a thread-safe handler in the federate ambassador. The later implementation was used for the experiments described later.

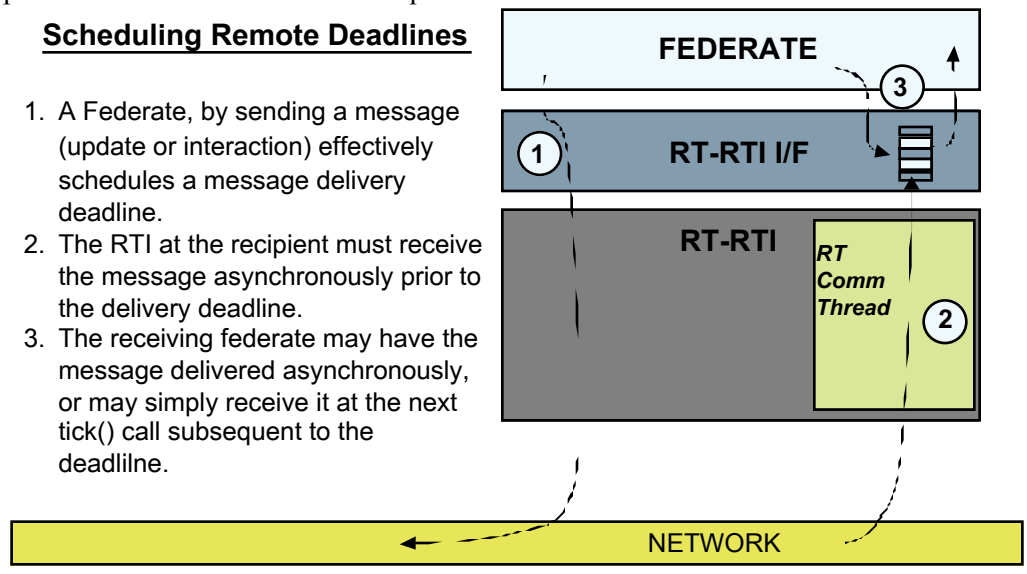


**Figure 3 - Real-Time Delivery Assurances**

### 4.2. Real-Time Service Assurances

The current HLA IFSpec makes no guarantees with respect to the timeliness of services. That is, when invoking an RTI ambassador service, the federate has no control or assurances regarding the amount of time it takes to complete the service request. In a real-time execution, scheduling

processes to meet deadlines depends on predictable bounds to execution time. From experience, we observe that many RTI services return immediately and are very predictable. However, certain services may require a distributed operation and will not return before communicating with other processes.

For example, subscribing to an interaction class, or object class attributes, might require modification of the communications configuration that implements a federation multicast mechanism. Depending upon the communications implementation, and the federation execution itself, the amount of time required to complete the subscription request can vary greatly. In some implementations, the RTI may return from the request immediately, even though the request has not been handled by all *local RTI components* (LRCs). This means that the federate has no assurance regarding when it will actually begin to receive the messages to which it subscribed. Another example of single-service assurance needs is in data distribution management. At present, there are no guarantees regarding the amount of time required to perform region subscription, modification, or deletion. An approach to synchronized DDM was presented in [8]

Figure 4 depicts the process of managing a distributed service invocation to return by a real-time deadline. This architectural approach depends on a real-time service handler, which can respond to local and remote service requests asynchronously. Intuitively, a full implementation of this approach necessitates real-time operating system functionality. RTI services must be scheduled locally to meet local and distributed deadlines.

In addition to single-service assurances, we can see the need to provide assurances regarding service sets. Ownership transfer, for instance, may require several requests and responses. Providing bounds to an ownership transfer process could be very useful in a tightly coupled behavior where it is advantageous to consolidate the processing of the objects involved on a single federate.

Time management is another service set that could benefit from real-time assurances. In non-real-time execution, when a federate makes a time-advancing request, there is no guarantee when (or if) a corresponding time-advance grant will be received. In a real-time execution, however, the relationship between local simulation time and WCT might be exploited to provide a bound for a time-advance grant. This concept is explored further in Section 6.2

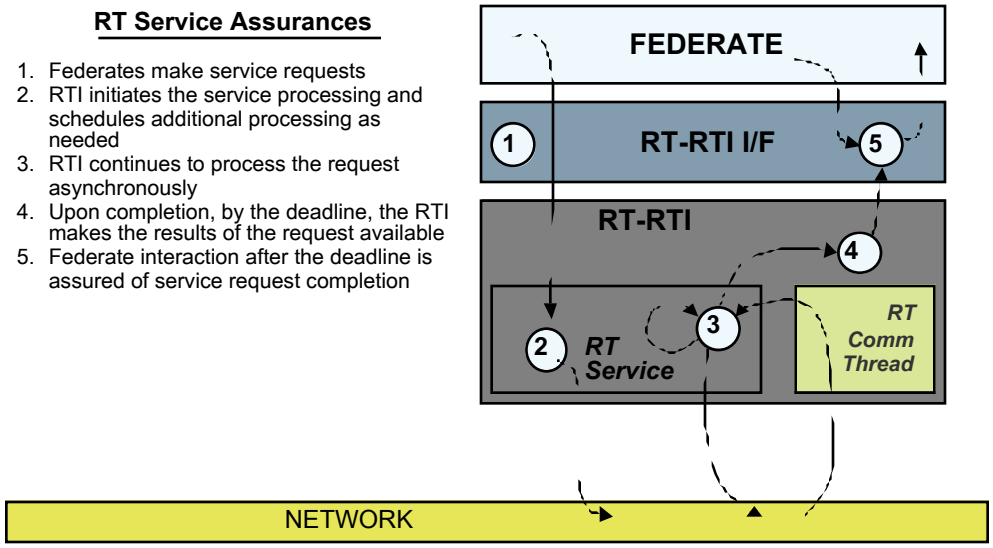


**Figure 4 - Real-Time Service Assurances**

It is important to note that some service assurances may depend on the independence of RTI and federate processes. Although it might be possible to make such guarantees by placing restrictions on a federate execution model (to ensure timely RTI processing of incoming messages), conceptually, managing RTI processes in real-time is a system level activity, and should not depend on federate, application-level intercession. Practically, this implies some degree of RTI asynchrony with the federate thread.

### 4.3. Fully Distributed State Assurances

In this architectural concept, the concept of asynchronous RTI services is extended to include all HLA services that can logically be RT bounded. We can consider such an HRT RTI as providing a distributed simulation real-time virtual machine as shown in Figure 5. The RTI, a system of real-time processes itself, would use a real-time scheduler to ensure that RTI services met the deadlines. RTI services could be modified to give deterministic results in real-time.

Using this RTI paradigm we might specify our federation execution parameters in very different way. Rather than specify a single value for lookahead, for instance, we might specify latency bounds for individual classes. The RTI could negotiate appropriate QoS guarantees to meet the bounds for a specified federation execution. Alternatively, we might choose to specify a federation time scale factor, SF, such that federation simulation time equals SF*WCT. We could choose to ignore the CPU clock in favor of a more useful real-time RTI-based federate clock. Message timestamps, in this paradigm, become simulation time scheduled delivery deadlines. Because of the RTI managed relationship between federation simulation time and WCT, the timestamp implicitly maps to a WCT deadline.

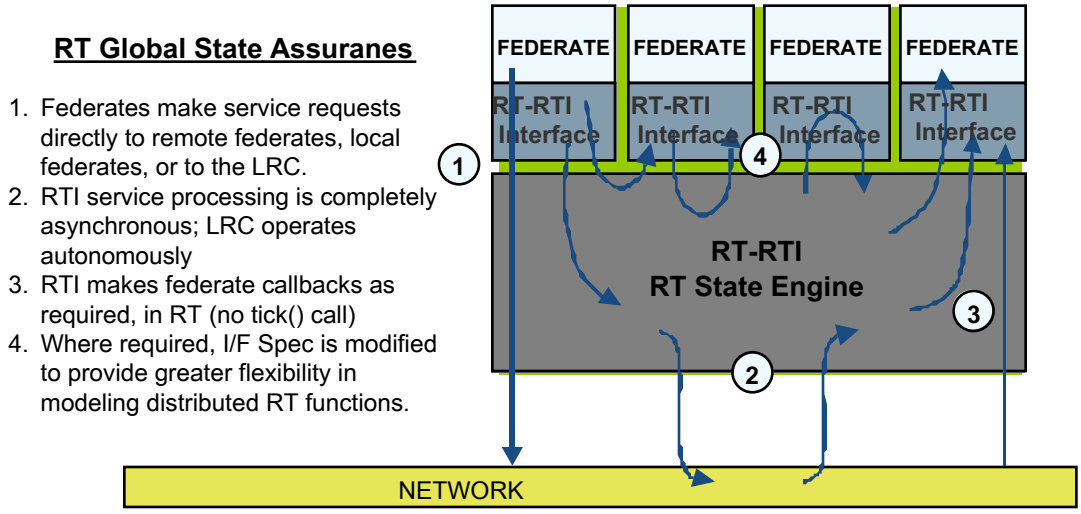


**Figure 5 - Real-Time Global State Assurances**

This paradigm also necessitates a shift in real-time modeling concepts. In a simulation where a federate is permitted to send messages (updates or interactions) timestamped with the current federate time, the federate is operating with *zero-lookahead*. Because the RTI must ensure that simulation time always relates to WCT, no zero-lookahead cycles of messages can exist in the system. Logically, this makes sense because a message cannot possibly have a scheduled delivery deadline equal to the time it is sent. In a real-time execution a zero-lookahead message is, by definition, always late. While we may choose to allow a late message in a soft real-time execution, a cycle of late messages can prevent the RTI from maintaining the relationship between federation time and WCT. This problem is one of the reasons for the prevalence of RO communications in current DS-RT systems. In rethinking the DS-RT modeling paradigm, a HRT RTI may offer more flexible ways to map message delivery requirements to network propagation properties, enabling more explicit control over the execution.

## 5. A Real-Time RTI Implementation

A real-time RTI was developed to demonstrate and evaluate the approaches discussed in the previous section, focusing on predictable performance with respect to message latency and time management operations. An HLA RTI based on the RTI-Kit software package was used as the starting point for this implementation, and was modified to include services with predictable

performance (assuming QoS guarantees in the underlying network). The original RTI-Kit software is described next, followed by a discussion of the modifications that were made to realize real-time performance. Specifically, the principal changes that were made for this study were to modify the design from a synchronous to an asynchronous message delivery mechanism in order to achieve more predictable communication latency, and development of techniques for real-time computation of LBTS values, the essential element needed to realize time management services.

### 5.1. RTI-Kit

RTI-Kit is a collection of libraries designed to support development of RTIs for parallel and distributed simulation systems. Each library can be used separately, or together with other RTI-Kit libraries, depending on what functionality is required. These libraries can be embedded into existing RTIs, e.g., to add new functionality or to enhance performance by exploiting the capabilities of a high performance interconnect. In previous efforts, the RTI-Kit software was successfully embedded into an HLA RTI developed in the United Kingdom [4]. More recently, the libraries have been used in the development of new RTI variants.

Multiple implementations of the RTI-Kit software have been realized targeting different platforms. Specifically, the current implementation can be configured to execute over shared memory multiprocessors such as the SGI Origin, cluster computers such as workstations interconnected via a low latency Myrinet switch, to workstations interconnected over local or wide area networks using standard network protocols such as IP.

RTI-Kit includes an RTI implementation called DRTI (Detailed-RTI) that conforms to the HLA Interface Specification (version 1.3). This implementation, included in Version 3.0 of the Federated-simulation Development Kit (FDK) was used here. Figure 2 shows a block diagram of the RTI-Kit.

Much recent work with the RTI-Kit has focused on new communications layer alternatives. This research is facilitated by the modularity of the RTI-Kit, restricting the majority of the changes to one or two RTI-Kit modules. A recently completed project replaced the existing communications module with an MPI implementation. A separate project demonstrated the RTI-Kit running over a wireless LAN on handheld (IPAQ) computers. Another ongoing collaborative effort is demonstrating the use of active-networks to solve difficult data-distribution problems [7].

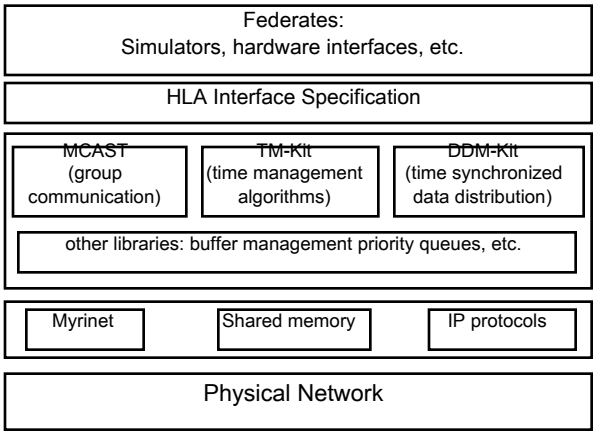


**Figure 6 - RTI-Kit Modules**

### 5.2. Asynchronous Communication

The communications layer of the original RTI-Kit software was redesigned in order to realize a real-time HLA RTI. In previous versions of RTI-Kit, network communication occurred

synchronously; messages were received directly from the communications hardware (the wire) at the request of higher layers (usually during the tick() call). However, this method requires the federate to pass control to the RTI in order to service the wire, which makes message delivery time, and RTI response time, inherently unpredictable. Therefore, our initial development has focused on asynchronous communication as a more predictable alternative.

Asynchronous communication provides the RTI with the ability to execute without interaction between it and the federate. The RTI-Kit MCAST module is responsible for providing subscription-based group communication. MCAST has been redesigned to work with ECho, a more flexible and powerful communications library developed by other researchers at Georgia Tech.

ECho is a collection of libraries that provides interfaces for network messaging and group communication [9]. It currently supports communication over TCP but is being extended to support additional transports (e.g., ATM and Reliable UDP) and QoS guarantees. ECho executes on multiple platforms including Linux, many other UNIX-based systems, and Microsoft Windows. Echo has powerful utilities to mitigate architectural differences in communication between any of the supported platforms.

Figure 7 illustrates the operation of the MCAST module, as modified to use Echo. MCAST initiates connections with other federates and manages communication between them. When a federate begins execution, the RTI-Kit initializes MCAST. Connection Manager (CM), an ECho library, provides a transport-independent API used by MCAST for unicast communication. During initialization, CM creates a thread to asynchronously receive messages and deliver them to the appropriate unicast handlers. ECho mediates group messaging through CM and passes group messages to the multicast handler.

The basic execution of the communication thread is intuitive. The communication thread suspends until a message arrives. Upon arrival of a message, the communication thread wakes, receives the message, and passes it to the appropriate handler. In most operating systems (e.g., Linux, Solaris, and Windows) the communication thread executes as a kernel thread. This allows it to block without blocking execution of the federate thread. In multiprocessor systems, both threads can run in parallel, allowing the federate to execute while simultaneously receiving messages. MCAST provides facilities to provide both synchronous and asynchronous handlers.

To exploit the asynchronous capability of the ECho/MCAST implementation, the DRTI was modified to provide for thread safety through the needed portions of the RTI, and ambassadors. The modified RTI provides asynchronous handlers for RO messages (i.e., objects and interactions). Therefore, RO messages are immediately passed up to the appropriate federate ambassador handler upon arrival.

This message delivery paradigm is analogous to interrupt-driven processes in which a handler is called following the occurrence of a defined condition. Asynchronous handlers are called from within the communication thread and do not depend on the state of the federate thread. It should be noted that unusually long handlers may interfere with the federate execution loop (this would also be the case with synchronous message delivery). Also, in this model, the federates are responsible for protecting data structures shared between the two threads. The thread-safety issues are unavoidable in a "tick-less" architecture.

In order to test the efficacy of this solution, a series of experiments was constructed to measure round-trip message latency under different federate loads. Experiments were conducted using two machines in a cluster of dual-Pentium II 300Mhz computers connected by a commodity 100Mbps Fast Ethernet network and running Redhat Linux 7.1. While the TCP protocol provides no guarantee on message delivery, an underutilized network will (in most cases) exhibit predictable behavior. We exploit this fact as demonstrated in the results.

All tests involve passing an object attribute of 128 bytes between two federates and measuring the round-trip-time (RTT) between message handlers. The performance of the asynchronous communications method was compared to a synchronous delivery method. The distribution of measured round trip times for this experiment is shown in Figure 8. The simulation load is simulated with a busy loop that executes in one of three modes. The first mode simulates no load at all; that is, tick() is called continuously until the end of execution. The second mode simulates a constant amount of processing between each tick() call and is similar to the execution of a time-stepped

simulation, as explained in Section 3.1. The third mode generates a load time drawn from a uniformly distributed random variable between each tick() call.

Each of the three tests was run for a series of 10,000 RTTs on the traditional synchronous DRTI and the asynchronous ECho-based DRTI. The tests were conducted on a relatively quiet LAN, with no special hardware. For the constant load mode, the amount of processing time was approximately equal to a 1 millisecond time-step.

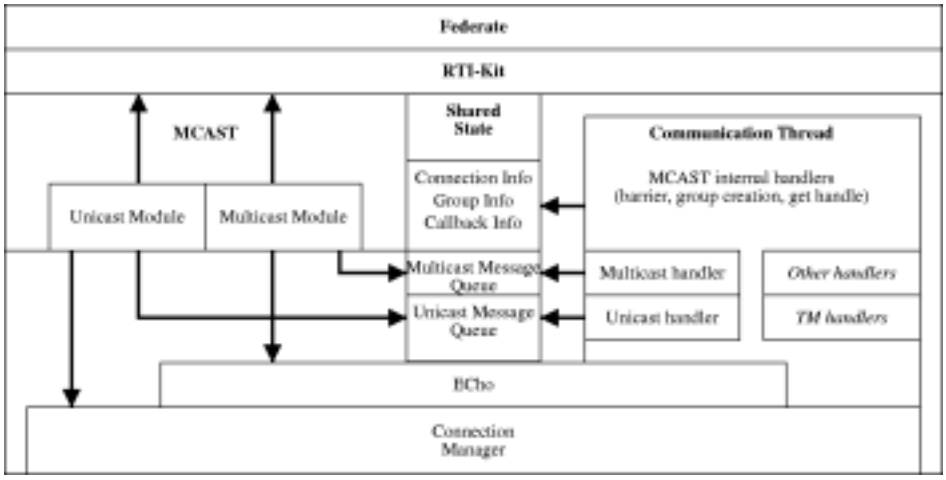


**Figure 7 - MCAST Modifications**

In Figure 8, one can see that the synchronous delivery with no-load yields the best possible performance. This is as one would expect, in that there is no computation interfering with the message passing. This mode is only useful as a nominal lower bound; it does not represent a useful model of a simulation. For the test network, we observed a minimum round trip time of 0.39 milliseconds, and an average of 0.43 milliseconds for the no-load mode.

For the constant-load and random-load modes, the RTT reflected the nature of the load. The random-load case shows a large flat distribution of round trip times. This is entirely due to the variable and unpredictable delay between tick() calls. There is somewhat more consistency in the constant-load mode. Note, however, the bi-modal behavior of the RTT delay. This is due to the likelihood that in some cases the messages do not arrive in time to be processed in one cycle of tick() calls. This behavior was also noted in [3], and is common for time-stepped execution models. The key observation in both of these modes is that low network latency, alone, is insufficient to ensure low latency interactions between federates. Delays caused by time-stepped execution, for synchronous delivery methods, will cause delays beyond network latencies. This is a significant motivation for asynchronous delivery in real-time simulation. For the constant-load mode, the lowest observed RTT was 1.05 milliseconds, and the average was 1.39 milliseconds. For random-load, the lowest RTT was 0.42 milliseconds, and the average was 1.097 milliseconds.

Finally, we see that for asynchronous delivery, the timeliness of the messages is very regular and predicable. The experiment was conducted for asynchronous delivery for all three load modes. However, because of the multithreaded implementation, and the multiprocessor test platform, the results for all three asynchronous load modes were virtually identical. As we would expect, the asynchronous communication method outperforms the synchronous modes. It is notable that there is a very small penalty associated with asynchronous delivery as compared to the baseline (no load) case. The minimum observed RTT for asynchronous delivery was 0.63 milliseconds and the average was 0.66. This gives us a practical upper bound for message delivery of well under 1 millisecond for this experiment.

As the RT implementation evolves, additional RTI services will register message handlers with MCAST to transition from deferred message handling to completely asynchronous execution of RTI

processes. Other RTI-Kit libraries such as the TM-Kit will be able to process messages without requiring the federate to give control to the RTI. As additional transports and QoS negotiation features are added to ECho we will extend the MCAST implementation and the RTI services to exploit the enhancements.

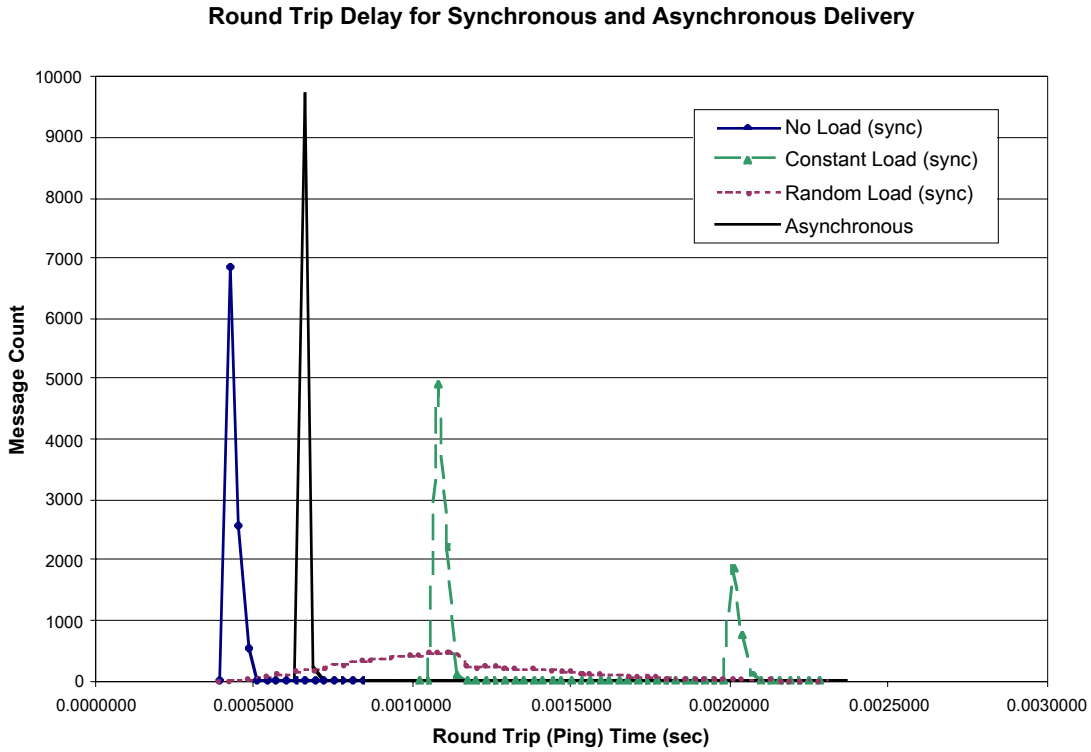


**Figure 8 - Communcations Performance Comparison**

### 5.3. Real-Time Time Management

Time management services provide a mechanism to order simulation events, e.g., to prevent non-causal event orderings. Specifically, each simulation message is assigned a time stamp, and the RTI ensures that (1) messages are delivered to each federate in time stamp order, and (2) no message is delivered to a federate in its past, i.e., the time stamp of any message delivered to a federate must not be less than the federate's current simulation time. The principal operation required to implement time management services is determination of the *lower-bound on timestamp* (LBTS) of any subsequent message that may be later received for a federate; this value is also known as the *global virtual time* in the context of optimistic time management algorithms. The LBTS value is crucial because any messages with time stamp less than (or possibly equal to, depending on details of the definition of the message delivery service) LBTS can be delivered to the federate while still guaranteeing time stamp order delivery. Further, so long as the federate's simulation time is not advanced beyond its LBTS value, one can guarantee that no messages will be delivered to a federate in its (simulation time) past.

In this section, we address the suitability of time management in a real-time RTI. The RTI implementation builds upon previous work in achieving repeatable real-time distributed simulations [3]. A key challenge in ensuring repeatable performance concerns the timeliness is the tick() function. Apart from administrative functions, the most significant source of overhead in tick() is

simulation time management.

In the original RTI-Kit implementation a distributed reduction computation is used to compute LBTS values. This cannot generally be guaranteed to complete within a bounded time because of two factors: 1) it depends on the participation of all other federates in the execution, 2) transient messages cause an LBTS computation to be aborted and retried. Here, the distributed computation was modified to compute LBTS in bounded time with a level of computational overhead that is appropriate for a RT execution. To ensure effective real time operation, one must also ensure that LBTS values are sufficiently large to permit delivery of messages in real-time; this is beyond the scope of this paper, however.

Using our previously stated assumptions, we have modified an LBTS reduction algorithm to exploit the latency bound and preclude transient messages. A transient message cannot occur if it is sent with a delivery guarantee prior to the beginning of an LBTS computation. Also, if the RTI time management routines execute asynchronously, one can use a real-time clock to schedule LBTS computations at a common time. Thus, each of the processors participating in an LBTS computation can rendezvous in real-time, and reliably and efficiently complete the reduction computation.

Results from performance tests indicate, as we would expect, that the modified LBTS computations are significantly more efficient than the original LBTS algorithm. Figure 9 shows execution results for an experiment conducted using a test federate with the original and optimized LBTS computations. For this experiment, the federation was run for 30 seconds of simulation time in a very coarsely synchronized cluster of Linux workstations, using TCP over Ethernet. With a reasonable upper bound to latency of around 2 milliseconds (actual numbers were much less) we created a real-time LBTS computation with a period of .25 seconds. Given an (arbitrary) value for lookahead of .3, we measured the efficiency of the RT-LBTS computation. Efficiency, in this case, is a measurement of the proportion of total simulation time spent in time management computation. The LBTS overhead factor, in Figure 9, is a ratio of real-time spent in the computation of LBTS to total simulation execution time. One can see that the overhead of the real-time approach is

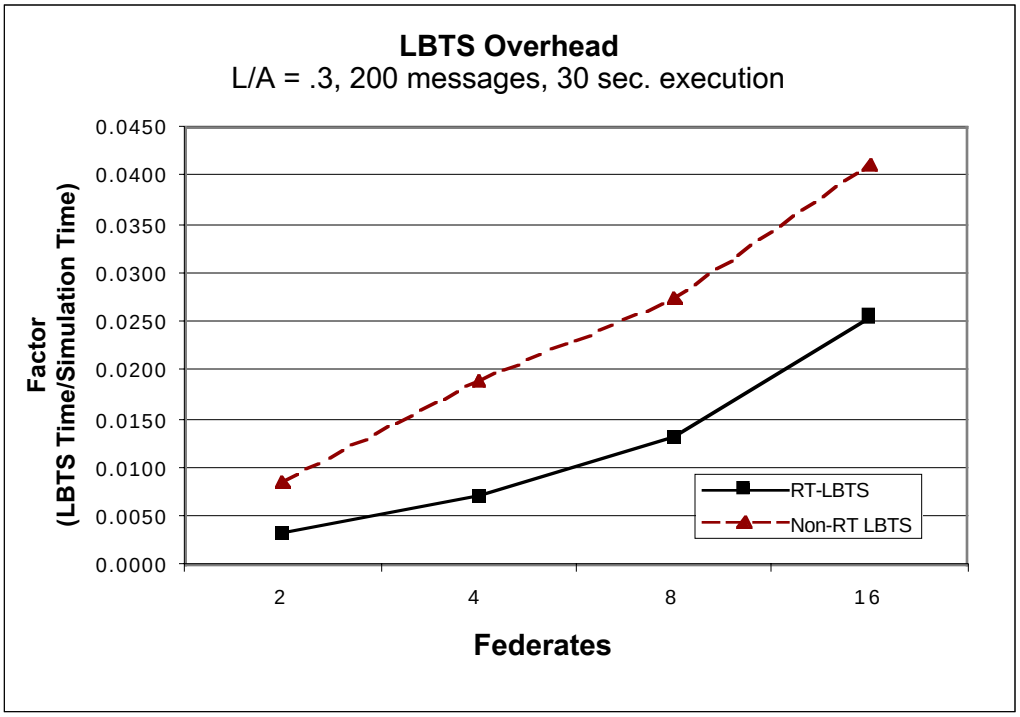


**Figure 9 - RT-LBTS Overhead Results**

significantly smaller than the original algorithm. This is due to the total number of LBTS computations and the number of restarts for the original algorithm. Whereas the RT-LBTS execution always resulted in 119 LBTS computations (the execution time divided by the LBTS period), the number of non-real-time LBTS computations ranged from 237 to 431 with up to 350 restarts. The apparent advantages of RT LBTS computations, even in this rough experimental configuration, suggest that hard real-time Time Management Services may be a way to meet difficult real-time integration challenges such as hardware-in-the-loop simulations or live test-ranges.

## 6. Real-Time Extensions to the HLA

The potential benefits of a HRT RTI cannot be fully realized if one is limited to the existing HLA IFSpec. Additional RTI configuration data or vendor-supplied specifications are needed for designing and executing in a HRT environment. The HLA does not include an execution model for real-time simulation. The HLA IFSpec does not provide sufficient detail to construct a real-time synchronization model. The communications transport and ordering parameters are not sufficient for mapping federation requirements into RTI performance requirements. Depending on federation goals, several types of extensions might be appropriate to address these issues including:

***- Extending RTI delivery specification to support QoS requests***

For real-time delivery assurances, the RTI should be able to determine a QoS requirement prior to sending updates or interactions. Using the current IFSpec, the RTI could use federate lookahead as a starting point. However, this quantity applies to any outgoing message. A better mechanism would allow the RTI to determine a range of QoS requirements based upon publisher-subscriber (producer-consumer) pairs. This would allow the RTI to manage delivery requirements across individual links, optimizing network utilization.

***- Specifying timeliness requirements for RTI and Federation Ambassador calls***

As mentioned earlier, except for tick() (or *invokeCallbacks()*, in the IEEE 1516) the RTI IFSpec is silent regarding the requirements for RTI responsiveness. Even the minimum and maximum time values for tick() are labeled as approximations (specifiable to at least 1 millisecond) with no absolute assurances. Practically, this makes it difficult or impossible to schedule RTI service calls to meet real-time deadlines. An alternative to specifying timeliness for RTI Ambassador calls might be to have each federate specify timeliness data for the Federate Ambassador calls. Using this configuration data a HRT RTI could act as a scheduler for federate processes.

***- Providing real-time synchronization requirements for RTI Service processing***

It is interesting to note that many RTI services are not synchronized with the management of simulation time as noted in [7]. This can result in non-deterministic execution even for non-real-time, conservative, time-managed federations. By stipulating RTI service requirements in simulation time, a HRT federation is able to deduce HRT RTI processing deadlines as well. Specifying RTI behavior in this way could be extremely valuable for test and evaluation or other environments where reproducibility of an execution is an important requirement.

***- Specifying real-time simulation execution models***

After many years of integrating real-time and interactive distributed simulation systems, many important problems of interoperability have not yet been resolved. By developing general and well-understood models for real-time simulation execution, one can begin to formally specify correctness criteria for real-time federations. Specifying real-time simulation execution models for the HLA may allow the next generation of RTIs to resolve integration difficulties faced today.

## 7. Conclusion and Future Work

A hard real-time HLA RTI will provide time-bounded services, and those bounds will be well-known to the simulation developer. This provides the developer with additional control over simulation behavior. It will also facilitate the design of more credible virtual environments in which relationships between real-time events are deterministic. Precise specification of simulation behavior and execution requirements can allow repeatable simulation execution, a first step in providing credible simulation execution environments for testing. Hard real-time execution can provide more

realistic virtual environments by specifying correct real-time behavior to reduce or eliminate temporal anomalies, making simulation based training much more effective than in current, non time-managed environments. Finally, specification of HRT execution can permit more effective use of communication and processing resources. As illustrated in the simple RT-LBTS experiment, a HRT simulation execution environment can reduce overall RTI overhead. Efficiencies in RTI may translate to greater scalability or fidelity in HRT environments.

Without changes to currently available RTIs, we cannot exploit QoS commodity networks when they become available, and the real-time functionality of the RTI will remain very limited. Further, design techniques for real-time simulations will not improve because existing RTI's cannot support superior real-time execution paradigms.

In this paper we suggest a more robust paradigm for hard real-time HLA-based simulation. We presented preliminary work on RTI enhancements to support hard real-time federation execution. Current work has focused on implementation of asynchronous communications and real-time LBTS computations. Experimental results demonstrate that this approach can yield RTI implementations with much more predictable performance properties. We expect that these RTI enhancements will provide significant benefits in designing real-time HLA federations. Future work will extend the communications module to manage QoS guarantees for the federation. Using communications assurances, and some facilities of real-time operating systems, we expect to make progress toward a HRT RTI.

This research raises additional questions regarding paradigms for real-time simulation systems. We expect to confront significant challenges in finding efficient scheduling algorithms for communication and processing that best meet simulation execution requirements. We also expect to uncover additional complexities in managing HRT execution. For example, in allowing federation specification of latency bounds, the RTI may have to account for federation execution requirements that violate the physical constraints of the system (e.g., the requested latency is not physically possible).

We expect the area of hard real-time simulation to provide a rich set of research topics. By developing and exploring new paradigms for hard real-time execution, we see real-time simulation as an increasingly important and credible tool for training, engineering, and research.

## 8. References


[1] Defense Modeling and Simulation Office, *High Level Architecture Interface Specification*, Version 1.3, 1998. Washington D.C.

[2] Fujimoto, R. et al, *Design of High-performance RTI software,* in *Proceedings of Distributed Simulations and Real-time Applications,* August 2000 (DIS-RT-2000*)*

[3] McLean, T. and R. Fujimoto, *Repeatability in Real-Time Distributed Simulation Executions*, in *Proceedings of the 14th Workshop on Parallel and Distributed Simulation, 2000.*

[4] Fujimoto, R.M. and P. Hoare, *HLA RTI Performance in High Speed LAN Environments*, in *Proceedings of the Fall Simulation Interoperability Workshop*. 1998: Orlando, FL.

[5] Ferenci, S. and R.M. Fujimoto, *RTI Performance on Shared Memory and Message Passing Architectures*, in *Proceedings of the 1999 Spring Simulation Interoperability Workshop*. 1999: Orlando, FL.

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

[7] Dorsch, M., and V. Skowronski, Modifying the RTI for Active Networks, in Proceedings of the Spring Simulation Interoperability Workshop. 2001: Orlando, FL. p. Paper 01S-SIW-007.

[8] Tacic, I. and R.M. Fujimoto, *Synchronized Data Distribution Management in Distributed Simulations*, in *Proceedings of the 12th Workshop on Parallel and Distributed Simulation*. 1998.

[9] Eisenhauer, G., The ECho Event Delivery System, http://www.cc.gatech.edu/systems/projects/ECho, 2000

[10] Gill, C., D. L. Levine, and D. C. Schmidt. The Design and Performance of a Real-Time CORBA Scheduling Service. Real-Time Systems, The International Journal of Time-Critical Computing Systems, special issue on Real-Time Middleware, 20(2), March 2001.

[11] Schmidt, D. C. and F. Kuhns, An Overview of the Real-time CORBA Specification, *IEEE Computer Magazine*, Special Issue on Object-oriented Real-time Computing, June 2000.

[12] O'Ryan, C. and D. C. Schmidt and J. R. Noseworthy, Patterns and Performance of a CORBA Event Service for Large-

scale Distributed Interactive Simulations. *International Journal of Computer Systems Science and Engineering*, CRL Publishing, 2001.

[13] Brutzman, Don, Zyda, Michael, Watsen, Kent, Macedonia, M. Virtual reality transfer protocol (vrtp) Design Rationale," Proceedings of the IEEE Sixth International Workshop on Enabling Technologies: Infrastructure for Collaborative Enterprises (WET ICE '97), Distributed System Aspects of Sharing a Virtual Reality workshop, June 18-20, 1997, at the Massachusetts Institute of Technology in Cambridge, Massachusetts, USA, pp.179-186.

[14] Brutzman, Don and Zyda, Michael "Cyberspace Backbone (CBone) Design Rationale," Proceedings of the 15th Workshop on Standards for DIS, Orlando, Florida.

[15] Watsen, Kent and Zyda, Michael "Bamboo - A Portable System for Dynamically Extensible, Networked, Real-Time, Virtual Environments," in the Proceedings of VRAIS 98, 16 - 19 March 1998, Atlanta, GA, pp. 252-259.

[16] Wuerfel, Roger, "A comparison of HLA and DIS Real-Time Performance," Paper 98S-SIW-042, Proceedings of the Spring Simulation Interoperability Workshop, Orlando FL, 1998.

[17] Zhao, H. and N. D. Georganas, "HLA Real-Time Extension" Proc 5th IEEE D S - R T ' 2001 Fifth IEEE International Workshop on Distributed Simulation and Real Time Applications, Cincinnati, Ohio, Aug.2001