Computer Science
Technical Report

Colorado State University

CORBA Based HLA/RTI Design Approach*

Klaus Schug
Anura Jayasumana
Dept. of Electrical Engineering
Colorado State University
Fort Collins, CO 80523-1373
Email: jayasuma@engr.colostate.edu
kschug@lamar.colostate.edu

Sandeep K. S. Gupta
Pradip K Srimani
Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873
srimani@lamar.colostate.edu
gupta@cs.colostate.edu

Technical Report CS-97-109

Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873

Phone: (970) 491-5792 Fax: (970) 491-2466
WWW: http://www.cs.colostate.edu

*To appear in the **Summer Computer Simulation Conference (SCSC '97)**, Arlington, Virginia, July 13-17, 1997.

CORBA Based HLA/RTI Design Approach*

Klaus Schug
Anura Jayasumana
Dept. of Electrical Engineering
Colorado State University
Fort Collins, CO 80523-1373
Email: jayasuma@engr.colostate.edu
kschug@lamar.colostate.edu

Sandeep K. S. Gupta
Pradip K Srimani
Computer Science Department
Colorado State University
Fort Collins, CO 80523-1873
srimani@lamar.colostate.edu
gupta@cs.colostate.edu

Abstract

DMSO's effort to provide a CORBA based, standard, common real time simulation architecture (High Level Architecture/Run Time Infrastructure, HLA/RTI) [1] for use in military, Civil Air Traffic Control (CATC - USA, Europe, Canada), the entertainment industry and NASA has not produced a CORBA based implementation with adequate real time performance. The features that are desirable in an RTI implementation are described. A common HLA/RTI implementation that is suitable for the above applications is one that: 1) meets real-time performance requirements, 2) is the same implementation for all federations or simulations, 3) is implemented using commercial-off-the-shelf (COTS) software and hardware, 4) does not alter or compromise the modularity or present design of the RTI, and 5) allows users and owners of simulation applications and systems to retain control of their systems by not placing RTI functionality in line with their application code.

Keywords: CORBA, Real Time Simulation, HLA, Interactive Simulation.

1 Introduction

The purpose of the HLA/RTI is to provide one architecture, one simulation framework and interface specification for all distributed simulations, whether FAA, Department of Defense (DoD), NASA, other government agency, or commercial industry based. In order to achieve this purpose, the HLA/RTI design and implementation must provide:

a. The required services and data exchange mechanisms for real-time distributed simulation

b. Real-time message (data) exchange performance

c. One common implementation for all simulation programs without necessitating custom software coding for each simulation

d. Modularity of design

e. Scalability to very large simulation exercises without simulation-by-simulation unique implementation modifications.

The HLA/RTI architecture is derived from the experience of past efforts to standardize simulation architectures such as SIMNET and DIS. This vast amount of simulation experience, represented by and accessed through the HLA/RTI Architecture Management Group (AMG) and Simulation Interoperability (formerly DIS) Workshops, has produced a HLA/RTI design that provides a, c, d and e: the required simulation services and data exchange mechanisms in a common, modular and scalable manner. An existing HLA/RTI implementation prototype is available that uses object oriented (OO) COTS products such as the Common Object Request Broker Architecture (CORBA) and IONA Technologies implementation of CORBA (ORBIX). The prototype fulfilled its purpose in providing a proof of concept and verification of the HLA and RTI design in the shortest available time.

The existing HLA/RTI prototype has been designed using CORBA and ORBIX in a distributed fashion. Different RTI modules run on different machines connected by a local area network (LAN) or wide area network (WAN). Although CORBA and ORBIX provide a natural implementation framework for the object-based HLA/RTI, the processing and communications load of such an implementation preclude meeting real-time performance needs of distributed simulation. The prototype does not meet real time performance requirements. As a result, an implementation of the HLA/RTI design that meets real-time performance requirements and also provides the desired design features is not yet available.

*To appear in the **Summer Computer Simulation Conference (SCSC '97)**, Arlington, Virginia, July 13-17, 1997.

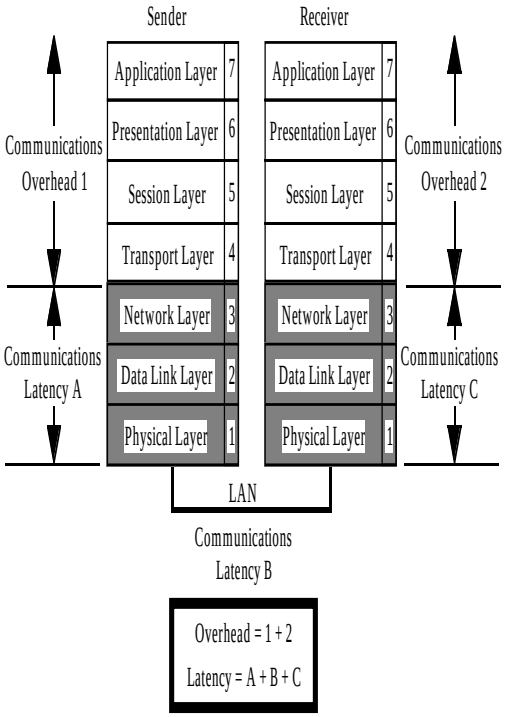

Figure 1: Communications overhead and communications latency

With respect to the characteristics of the RTI services, the performance of current CORBA and ORBIX is not yet suited for real-time simulations with constrained processing and communications overhead/latency systems. An OO implementation has increased processing overhead in contrast to traditional non object oriented or data oriented approaches. In order to provide the advantages of object oriented design, mainly easy to use and easily understood RTI services and interfaces, additional processing is required. This processing hides the object-orientation (encapsulation) and relocatable object platform-independence implementation details from the users. Long-chains of intra-ORB virtual function calls, and a lack of object oriented mechanisms integration with underlying operating system (OS) and network quality of service (QoS) mechanisms are additional sources of CORBA and ORBIX processing overhead. There is neither any priority mechanism nor are there any asynchronous requests to aid in speeding up processing.

2 CORBA Based RTI

An OO CORBA and ORBIX RTI implementation has a great deal of communications overhead and some additional communications latency in comparison to other non CORBA/ORBIX OO or non OO implementations. The time spent by the processor sending or receiving a message is defined as communications overhead. The time spent in the actual network hardware, in network interface units (NIUs) and in the network (e.g., transmission time in a local or wide area network - LAN or WAN), is defined as communications latency (Figure 1).

By examining the RTI architecture and functional block diagram illustrated in figure 2, by examining the HLA Interface Specification, and by examining the CORBA and ORBIX specifications, it can be seen that a great deal of communication is required between many RTI functions to perform RTI simulation services. If the RTI functions are in the same machine, then the communication requires processor time only and is therefore overhead. If the RTI functions are distributed in different machines, the processing of the functions is still required and additional network messages are required for coordination between the different machines, adding communications latency to the time delay.

A distributed RTI is a major factor to processing and communications latency/overhead. A distributed RTI means that each simulator participating in the simulation must perform some of the functions of the RTI. The degree of RTI distribution is determined by how many functions of the RTI must be performed at each simulator. With the main RTI functions depicted in figure 2, one recognizes that if there is a high degree of distribution, the RTI processing burden becomes a major performance impact. When many, perhaps all, of the RTI functions are performed at each simulator, then combined with the CORBA and ORBIX processing overhead, the processing required at all simulator machines can easily exceed the available capacity. The RTI processing load is particularly important for those simulators that were never designed to be networked and to which additional resources such as CPUs or memory can not be added. Even for newer, multiprocessor simulator machines, the additional processing and communication delays of a CORBA and ORBIX RTI implementation can cause the simulation application processing to fall behind to the point of simulation failure.

In a distributed implementation of the RTI, where subsets of RTI modules run on a cluster of simulators connected by local area networks (LANs), a close interaction between different RTI modules is needed to ensure the timestamp ordering of message delivery. Hardware and software LAN technology was not initially developed for fast and frequent communication between tightly coupled functions (as in parallel processing), and thus the communication overhead between networked simulators can be quite high. A CORBA and ORBIX implementation, when combined with a distributed RTI implementation, provides a number of other inherent processing and communications delays and overheads:

a. Non-optimized presentation layer conversion and excessive data-copying

b. Internal message buffering mechanisms leading to non-uniform latencies

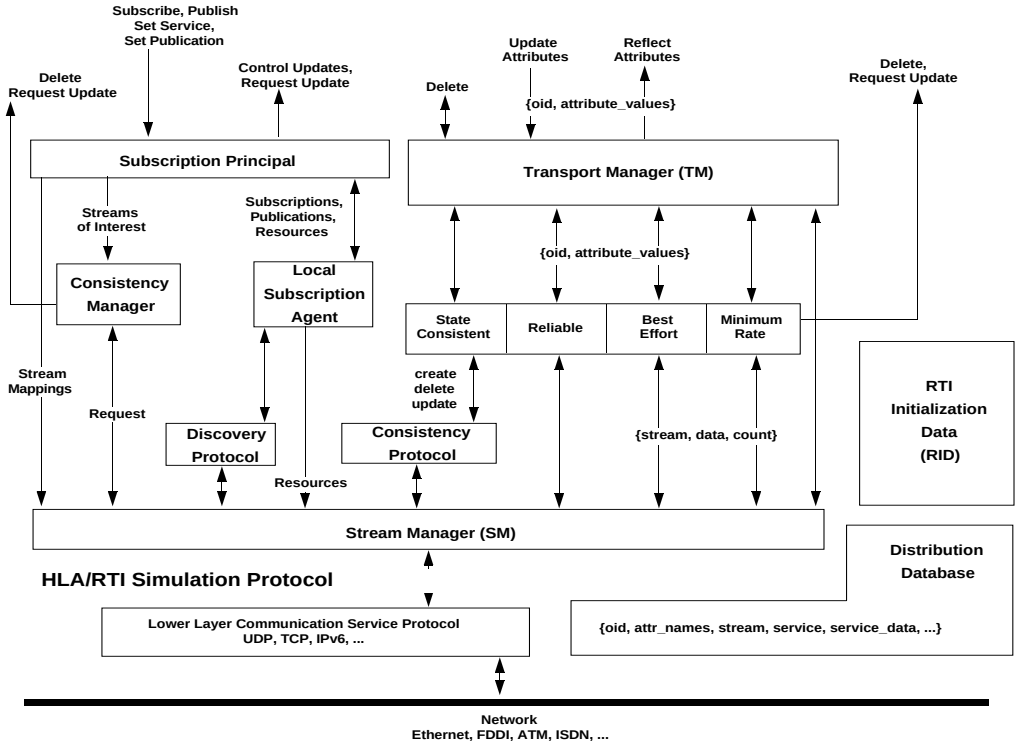

Figure 2: The RTI has many computationally and communication intensive functions

c. Inefficient receiver side demultiplexing and dispatching

d. Synchronous stop-and-wait flow control

e. Non-adaptive retransmission timer schemes.

Efforts to speed up RTI distributed simulation communication often include placing the RTI functions code in line with the simulation application code, making for one large simulation application process. This implementation is often not obvious from the design descriptions, block diagrams and engineering reviews. The situation results from the need to speed up RTI processing as much as possible without sacrificing needed functionality. Once all unnecessary RTI functionality has been discarded and all remaining functionality has been combined in the most processing efficient manner, there remains one other "simple" speedup option for a distributed RTI: place all RTI and other communications function code in line with the existing simulation application program code. This will produce significant speedup of RTI execution, but at a price usually not made clear to the owners of the simulation application programs or the systems running the applications. By placing all RTI and communications functions (anything above the TCP/IP protocol) in line with the application code, the owners and users of the simulation applications and their systems lose all control over their applications and systems. Users and owners are not allowed to modify their systems and simulation applications without third party approval even for stand-alone, internal organization simulations. Figure 3 illustrates this situation. Such a loss of control often results in frustrated users duplicating their simulation applications on a new, stand-alone system in order to be free from third party RTI operations and maintenance restrictions. Duplication of the simulation systems as a result of a loss of control by the uses and owners of such systems, defeats the entire purpose of networked simulators and increases the cost and maintenance of simulations beyond the cost of pre-networking. Duplicate systems are procured, two independent versions of the simulation applications must be maintained, and additional personnel are required to maintain the RTI in line code and duplicate applications and systems.

3 Centralized RTI Design

Our design (figure 4) provides user and owners of simulation systems with the best method of maintaining control over their applications and systems. Our design minimizes the RTI code at each simulator, therefore not requiring that the RTI code be placed in line with existing application code. Not enough RTI functionality is required at each simulator to warrant integration with the application. No speedup at the simulator is required. Indeed, with simulators already containing network distributed simula-

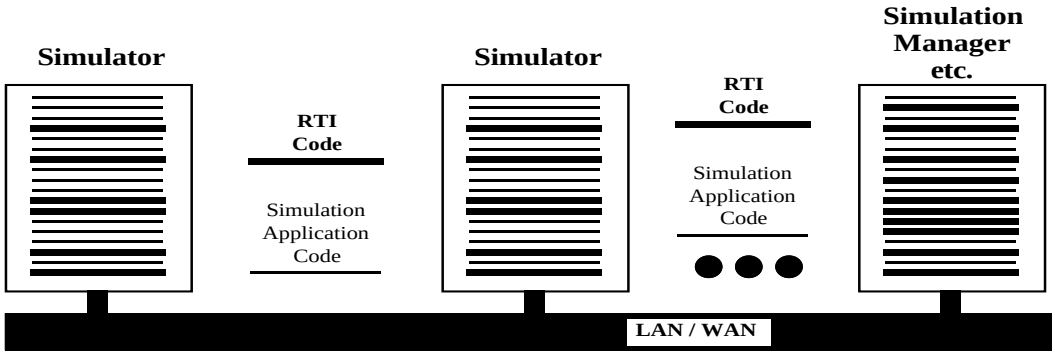

Figure 3: RTI speedup often "silently" places RTI code in line with application code at a horrific user impact and cost

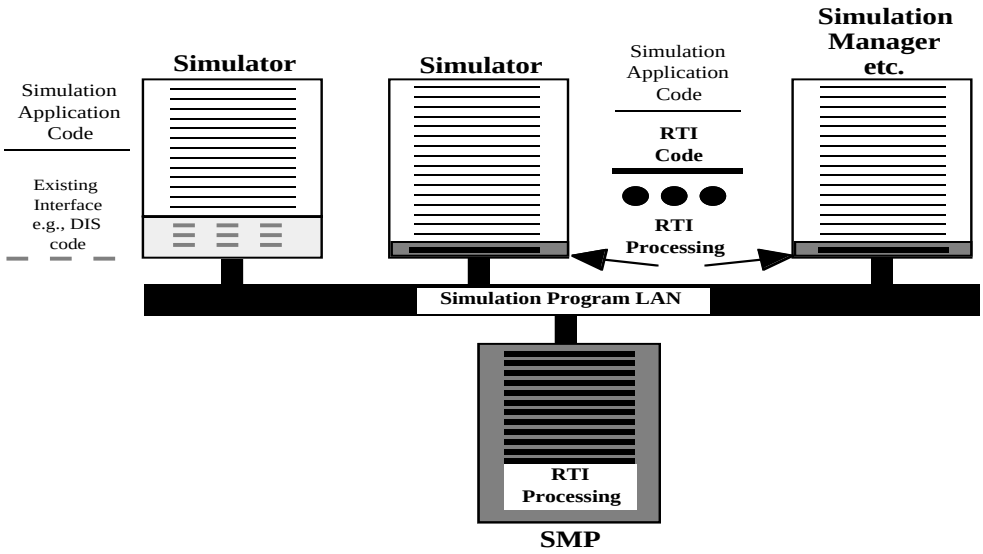

Figure 4: A centralized RTI maintains user control over user applications/systems

tion communications, the existing protocols and message sets can be used with the centralized RTI node making any necessary transformation. In such cases, no code needs to be added, either in line, or as separate processes invoked through context switches between simulation application, OS, and RTI processes.

In order to meet the real-time constraints on message delay and avoid the horrific cost of placing RTI code in line with simulation application code, RTI functions must be implemented as separate processes, executed more quickly, with less processing and communications overhead at each simulator, and executed with less communications latency for the entire simulation network. Our design and implementation provides all of these characteristics.

4 RTI Implementation

As discussed in the Introduction section, poor RTI performance is due mainly to communications overhead and processing. In our design, all the RTI services, functionalities and modules are implemented on an inexpensive, COTS SMP machine. Centralizing the RTI implementation to a high performance, parallel processing machine improves performance for the following reasons:

a. Minimizes the amount of RTI processing required at each simulator

b. Minimizes the amount of network bandwidth (messages) required for intra-RTI messaging

c. Executes RTI functions in parallel

d. Minimizes the amount of memory accesses required for RTI execution

e. Reduces (in many cases eliminates) all intra RTI module communication overheads considerably since all interactions can be implemented using uniform data formats on shared memory

f. Performance impacts from increased simulation size (more entities, etc.) are limited to the central RTI server

g. RTI maintenance and modifications are limited to the central RTI server.

Poor performance of a distributed OO RTI can be mainly attributed to communication latency and overhead incurred in exchange of information between different entities. Factors such as propagation delay and network interface delay that define latency, depend on the technology and the physical network configuration and therefore can be influenced only marginally. However, communication overhead, caused by the processor cycles required to deal with the information at each node, is more important. Response times seen by an application are dominated by the communication overhead. By centralizing the RTI functionality, the communications and therefore processing overhead at each node is minimized. The central RTI machine performs most, if not all, RTI module communication within itself. This minimizes any RTI network communications, reducing the number of messages traveling over the simulation LAN/WAN, reduces the number of messages that each simulator node has to receive and transmit, and minimizes RTI induced processing at each simulator. These reductions in network bandwidth, and the number of messages and RTI code that simulators have to process, has tremendous performance gains. In addition, a perhaps even greater benefit results. With the reduction and perhaps elimination of any additional RTI processing, functions or communication messages, simulators that could not accommodate any further processing can now be used as is within a distributed simulation. No recoding, communication front ends, rehosting or resource additions are required. In addition, if the simulator already uses DIS or another protocol to communicate, that existing protocol and interface can be reused for HLA compliant simulations by making the conversion to RTI services only once, at the centralized RTI node. Figures 5 and 6 illustrate the design principles.

Several other reasons why a centralized RTI implementation will be more efficient are as follows:

a. Multicasting is most efficiently implemented as the centralized version will have all the information needed to make the decisions

b. Data Logging is the most thorough and would have a single, common format output file.

c. Overhead required to make sure that the different RTI entities have consistent state information will be reduced

d. Overall network traffic is reduced as the need to handle multiple acknowledgments, etc. will be less

e. SMP machines can achieve very high effective memory bandwidths, reducing the biggest bottleneck to communications overhead - memory access, thereby greatly speeding up RTI function execution.

It is to be noted that all RTI functions may not be able to see the same level of improvement when centrally implemented. One interesting question is to categorize RTI services into groups that may have different levels of effect of centralization or distribution. For example, if we consider Time management services of the RTI (these services coordinate the advancement of logical time and its relationship to real-time during the federation execution), we see that a centralized RTI would be suitable for simulations whose requirements for time management either fall in Category I (paced with agreement) or Category III (Not paced with agreement). The RTI manages the current federation time by arbitrating between the Time Advance Requests of the simulations in the federation. The RTI grants a time advance to the simulation(s) whose requested time falls next in sequence. On the other hand, centralization of RTI implementation may not have much impact on simulations which do not require the RTI to play an active role such as in Category II (paced with no agreement) and Category IV (not paced with no agreement) simulations. Our implementation efforts and testing provides an exhaustive experimental measurement going a long way to give new insight into fine tuning the implementation strategies to optimize performance over all kinds of simulation clients.

Efficient parallel execution of RTI subtasks is necessary to meet the real-time constraints on message delay. Efficient parallel execution is often dependent on a high level of synchronization between tasks. There is typically little support for this in the existing complex, multitasking, multiuser simulator environment. A centralized SMP based RTI provides efficient parallel task execution support hardware and software mechanisms reducing context-switch overheads.

Unlike DIS, the RTI presents data in format native to the processor. This incurs extra overhead in the form of multiple memory accesses and system data bus transfers which reduce performance. SMPs have multiple high-performance multiprocessors that (logically) share the same memory. Since they provide a single address space, shared-memory multiprocessors are a convenient platform for both serial and parallel programs. They are also cost-effective for high-performance computing since they take advantage of commodity microprocessor and memory components. Most major computer manufacturers offer relatively inexpensive SMP systems.

5 Conclusions

In this paper we have presented a HLA/RTI implementation approach which will meet the real-time performance

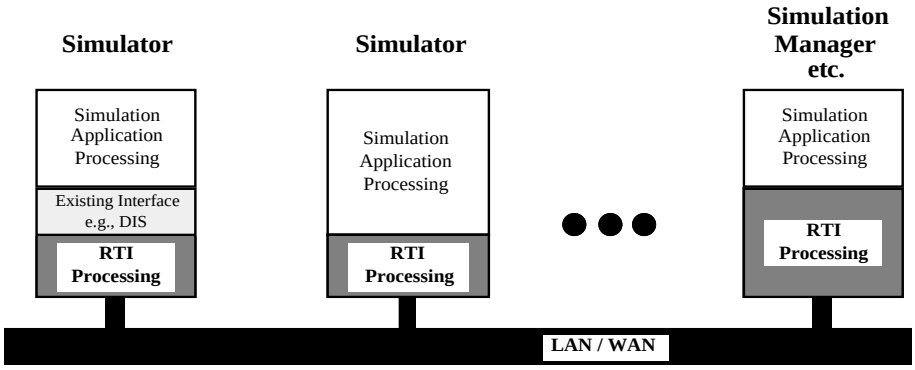

Figure 5: A distributed RTI causes extensive processing and communications overhead at all simulators in the simulation exercise

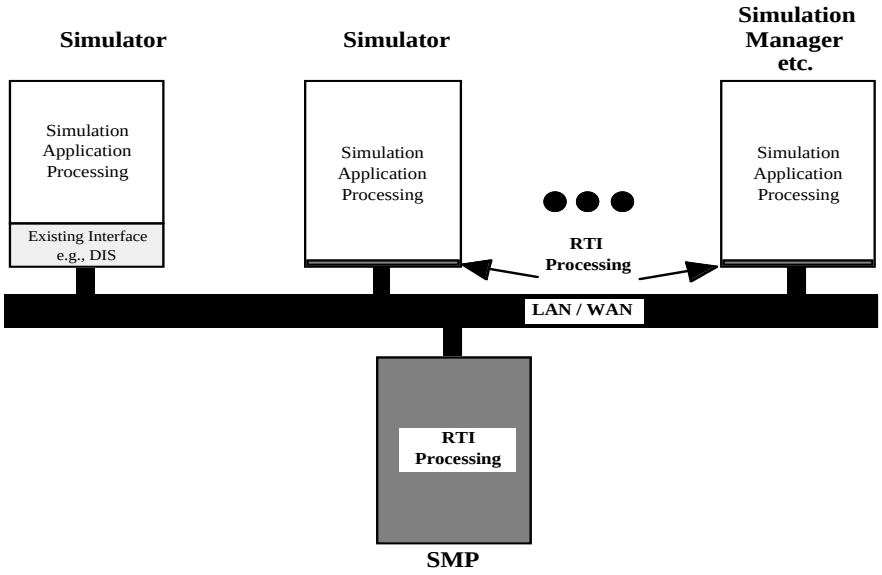

Figure 6: A centralized RTI minimizes processing and communications overhead at all simulators in the simulation exercise

requirement of distributed simulations as well as be scalable to STOW-size simulations.

References

[1] DMSO 1997. HLA interface specification version 1.1, final draft, february 4, 1997. Available through internet http://www.dmso.mil/projects/hla/, January 1997.