Proceedings of the 1999 Winter Simulation Conference
P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, eds.

NETWORKED SIMULATION WITH HLA AND MODSIM III

Glen D. Johnson

CACI Products Company
3333 North Torrey Pines Court
La Jolla, CA 92037, U.S.A.

ABSTRACT

This paper describes a networked simulation application using HLA (The High Level Architecture) and MODSIM III, a commercial off-the-shelf (COTS) object-oriented simulation language.

The Department of Defense (DoD) developed HLA for training simulation exercises, but HLA is applicable to a wide range of simulation work far beyond wargames. HLA is documented in terms of C++ and Java while Discrete-Event Simulations are often developed in a simulation language such as MODSIM III, SIMSCRIPT II.5, or SLX, or by using a graphical, domain-specific simulator such as COMNET III, SIMPROCESS or ProModel.

The requirement addressed by this paper is to interface an existing Discrete-Event Simulation model to the HLA, in order to evaluate that task and to set directions for future work. To further direct focus on interfaces between HLA and a Discrete-Event Simulation, we deliberately chose a small simulation application developed in MODSIM III. Real simulations are, of course, more detailed, but will still use the same interfaces.

1 INTRODUCTION

The way that computing is done has changed. The industry has moved from an application-centered style of computing into networked computing. There are many protocols available for networked computing and it is even possible to invent a new protocol for each new application.

At the same time, the DoD has made significant investment of research effort in the area of distributed simulation. The current evolution of this work is the HLA. This supports a vision that simulations can be constructed from reusable components that inter-operate through an open systems' architecture. There is a policy mandate that encourages the use of HLA for all DoD simulations, and the HLA is going through the standardization processes.

The HLA is described quite well in the literature (Dahmann 1998). Terms such as Federation, Federate, FOM, SOM, and RTI will not be defined yet again here. Readers unfamiliar with these terms may find them on the following Web page: http://hla.dmso.mil.

Developing a new protocol for each application is no longer necessary. Using a standard such as HLA encourages interoperability and should simplify the development process.

CACI Products Company's vision is to make simulation models operate together through computer networks, enabling simulation analysts to easily use networks. The High Level Architecture was investigated as one of the vehicles that could make this happen.

To do this, CACI built MODSIM III language bindings to the HLA standard. In order to provide the best possible support for HLA, the MODSIM III interface is integrated with other parts of the MODSIM system such as SimGraphics and provides a level above the HLA to better support message pumping, exceptions, and callbacks.

MODSIM III with HLA support also provides universal data value representation so that data can be seamlessly transported "on the wire" between different computers and operating environments. Since MODSIM III with SimGraphics is available on Windows 95/98/NT as well as on all popular Unix Systems, it is easy to load-balance a simulation model through a network or to display the user interface and animated graphical output on different or remote computers.

2 DISCRETE-EVENT SIMULATION

Discrete-Event Simulation is significantly different from other computer applications. Typically, simulation is used to explore possible outcomes. A simulation model is often a prototype that evolves as more is learned about the system being studied. The insight gained from modeling can be just as important as the results of running the model.

Effective Discrete-Event Simulation incorporates detailed knowledge of the area being simulated. Such knowledge is the province of people who have invested much of their career mastering the subject area; they are often not expert in computer programming. In order to

make the simulation technique even more accessible, the computer scientists should focus on providing tools to be used by experts in the subject area.

Most Discrete-Event Simulations include queuing, and studying queue behavior is often the primary objective of the simulation. Statistics collection is important, whether the results are displayed as tables of numbers or in graphical form. A good reference on Discrete-Event Simulation is (Marti 1999).

Random number generators are often used to simulate behavior that is "outside" the area being modeled. Random number generators are also used to model processes whose detailed behavior is not relevant to the study or whose details are at too low a level to contribute to understanding overall system behavior. Monte Carlo techniques are a very useful way to abstract real-world behavior into simulated behavior.

We want to carefully distinguish between a Discrete-Event Simulation and an Emulation. Too often, simulation studies get bogged down by attempts to provide motion picture quality graphical detail. And too often, studies attempt to model irrelevant detail. Deciding which details to model can be made much more scientific by stating explicit objectives. The specific question to be answered dictates the required details. The need for judgement as to which details to model means that simulation is as much an art as a science. For further information, refer to (Page 1998).

Discrete-Event Simulations are time-based, and a queue of pending and future events is the nucleus of any simulation. Rules for breaking ties, when two things could happen at the same precise moment of simulated time are often very important to the simulation scientist.

In a Discrete-Event Simulation, when any event runs, it has visibility and access to much, if not all, of the system state. And the change in state that an event provides is assumed to happen all at once. This instant visibility simplifies the simulation.

Simulation languages have invested much effort in efficient event management. Much of the data for queues and an event loops is cached by a modern single processor computer and this further enhances event selection and dispatching performance.

However, modern computing is now done across networks rather than on isolated single processors. Research work into techniques for Parallel and Distributed Simulation, represented by the HLA, can be used immediately in Discrete-Event Simulation, even though networks are usually much slower than computer processors.

The most obvious targets are simulations requiring multiple replications for statistical reliability. These can be done in parallel. Other models may explore multiple branches in parallel, playing what-if scenarios to find the optimum path.

Some models are best operated in a distributed way. They may contain interactions with physical components or data sources that are not all at the same place, and output may be required in distant locations. For these models, we can talk about "peel-off" inputs and outputs, the notion being that the model may run separately from its user interface.

New models can always be developed in the traditional way. But simulation models can also be built in components that can inter-operate through networks.

3 METHODOLOGY

Discrete-Event Simulation projects often use a rapid methodology, starting with a rough model and adding refinements and improvements as results unfold. HLA documentation suggests a methodology that seems most suited to interfacing between projects being developed by cooperating agencies, but may be too cumbersome for experimental Discrete-Event Simulation work.

Methodology is important. But, for Discrete-Event Simulation, results can often be obtained quicker and more cheaply by deferring or omitting many of the formal documents. Early results often are enough to decide whether an approach should be pursued, abandoned, or substantially modified.

A valid HLA Federation has an HLA FOM File and it is a set of federate programs that exchange data through the RTI. For a federate to be reusable as part of other federations, there must also be a SOM File and the documentation provided by the HLA OMT.

The information in the SOM file and the HLA OMT documentation is repeated in MODSIM III Definition Modules. As an aid to maintaining the match between this information, the language tools may, in the future, be extended to generate one from the other.

For exploratory work, the HLA interface can be developed by drawing an Abstract Model (Figure 1) on a whiteboard and then manually deriving the FOM from the whiteboard. At a later stage, the model can be formalized with tools such as the HLA OMT. If the Abstract Model is too large for the whiteboard a higher level of abstraction may be most appropriate.

Once there is an abstract model, it can be partitioned into Federates and the code for the simulation model and its interfaces can be provided.

The material needed for the HLA FOM is a static snapshot of the data world of the simulation, it is not enough to completely capture the behavior of a system. Other tools, such as State Transition Diagrams or diagrams from the Universal Modeling Language (UML) (Booch 1999) will be needed if the intent is to communicate the design to team members, customers, or model users. The details of the approach always depend on the skill, experience, and desire of the simulation modelers, this is another part of the art of simulation.

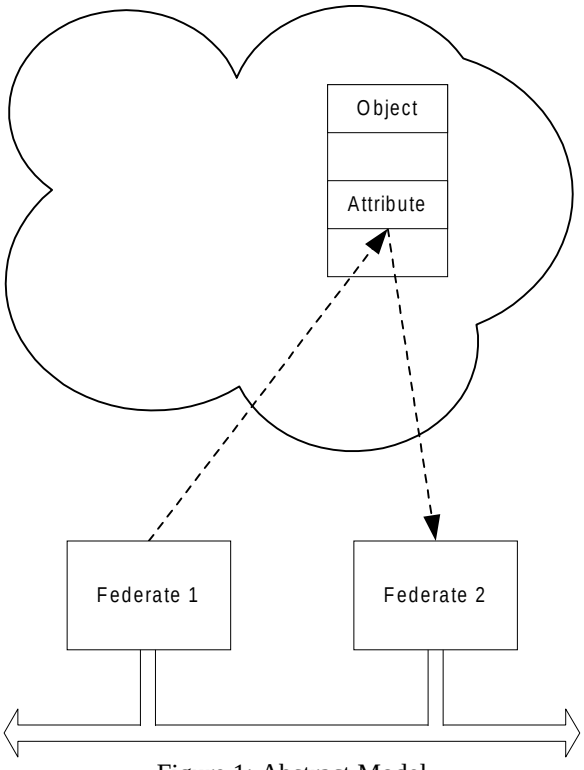

Figure 1: Abstract Model

4 MODSIM'S HLA INTERFACE

4.1 Data Interfaces

The DataSupport module provides the ValueObj, which allows representing any atomic data value or structure. This object can handle data sizes and data types found in C++, Java, SIMSCRIPT II.5, and MODSIM III applications. This object also takes care of byte order sequences (endian issues) found when moving between Intel X86 and various RISC-based computers such as the SPARC, MIPS, and Hewlett-Packard.

4.2 RTI Function Support

The MODSIM HLA interface provides an API corresponding to IEEE P1516.1, the current IEEE Draft Specification for HLA. At the current time, the implementation uses the C++ bindings to RTI 1.3V6 and accompanying documentation.

4.3 Exception Mapping

The RTI uses the exception feature of C++ to communicate return status information. The MODSIM III interface to HLA maps these to and from enumerated return codes.

4.4 Representing Time Values

Simulated time is handled in MODSIM III as a unitless REAL value stored in 64-bit double precision floating point format. Typically the value is in seconds, though other units are possible. The same notation is used for both simulated time values and time intervals.

The RTI::FedTime class of the HLA RTI 1.3 represents time as a set of pure virtual operations wrapped around a federation-specified byte sequence. RTI 1.3 provides one concrete realization of this class, RTIfedTime, that represents time values in C++ double floating-point. RTI::FedTime objects can represent either logical time values or time intervals.

The interpretation of time values is the responsibility of the Federation. One federation could represent its time as a 32-bit integer while some other federation could represent its time as a coded GMT character string. Yet a third federation could represent time values as a record where one field might serve as a priority value.

4.5 Representing Federation Time

The MODSIM III binding to the RTI represents, by default, time values as MODSIM REAL values. These are the natural notations for a MODSIM simulation.

4.6 Passage of Time in MODSIM

A MODSIM simulation model has a single clock; this clock sequences operation of the entire simulation model. The clock starts at 0.0 and runs forward for the duration of the simulation. Future activities are kept in a queue ordered by absolute time. Activities that are waiting for resources are kept in separate queues.

SimGraphics provides animation features. MODSIM supports this by using the value of variable Timescale as the number of wall clock seconds per simulated unit. real-time animation is optionally supported by skipping graphical updates.

4.7 Time Management and the HLA

HLA Provides general interfaces between a variety of schemes for managing simulated time.

The major issues in time management are network latency and the actual wall clock time it takes to synchronize clocks across a network.

Discrete-Event Simulation typically has performed best when simulation interactions happen within the same thread. Crossing threads takes at least 10 times as long, and changing address spaces, to a different process (or processor) is yet another 10 times slower. Simulation interactions across a network will add another order of

magnitude to this, meaning that the interaction between two simulated processes across a network may take 1000 times as long as entirely local interactions.

It is tempting to levy the developer of the simulation application with responsibility for synchronizing time across a network, under the view that a serious time aberration is a flaw in the modeling. Since the developer of a simulation already has a daunting task, it is possible that this new responsibility will not be welcomed.

5 JOB SHOP CASE STUDY

For the proof of principle, the MODSIM III implementation of the Classic Job Shop model (Figure 2) was chosen. This consists of one main module, with 306 lines of code and comments. The model demonstrates the use of ResourceObj and QueueObj, and uses a TELL method and the WAIT statement to perform a process-based simulation. Statistics are collected using features of MODSIM's StatMod.

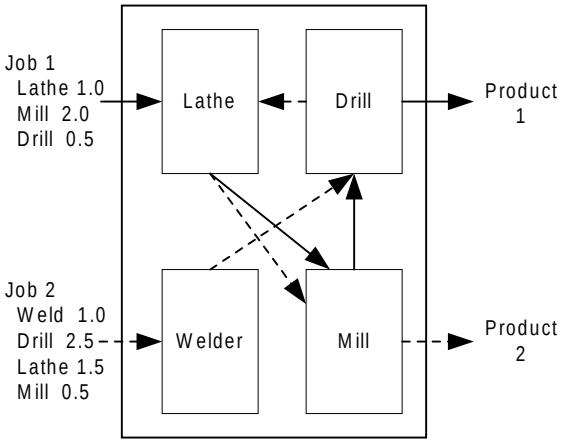

Figure 2: The Job Shop Model

A Job Shop is a system that contains a number of machines arranged in groups of identical machines. Jobs are orders that come into the shop. Each job is modeled as a sequence of tasks that must be performed sequentially, and each task may require a different machine and has a separate duration. During the run there may be times where there are not enough machines for the workload, so jobs are queued as necessary. The simulation model produces periodic reports on job and queue statistics. The reports allow the simulation modeler to experiment with system configurations to evaluate system performance. Simulation of this model is described in (Russell 1983).

The Classic Job Shop represents a large class of Discrete-Event Simulation Models. By changing the input data, it can become a communication model, a transportation model, or a supply model. However, even though this Job Shop is a complete simulation application, the Classic version does not use a network.

In order to develop an interface between the Job Shop and HLA, the simulation model was extended with the concept of field sales offices. Orders generated by Field Sales Offices come in to the Front Office of the simulation and are processed as jobs. The periodic reports are returned to the field sales offices.

To support the extensions, the implementation of the Job Shop was restructured into an HQ module, a Sales module, and a minimal MAIN module. The HLA interface was implemented in a Common module and a Networked module. This program is small enough that the same executable can be used for all federates, with its precise function being set by a run-time parameter.

Adding the Front Office and the Field Sales Office code, and adding the associated definition modules brought the application to 760 lines of code and comments. With the HLA interfaces, this increased to 2889 lines, including trace and debugging code.

The Job Shop case study is being used to demonstrate and test the MODSIM III interface to the RTI, on both Windows and UNIX, and to evaluate improvements. We are moving the HLA interface code into the MODSIM HLA support library, using mix-in inheritance to reduce the amount of code that needs to be written by the application developer.

6 RUNNING THE CASE STUDY

The case study was first run on Pentium-class computers running Windows NT 4.0, with RTI 1.3V5. The same executable can support both the Front Office federate and any Field Sales Office federates, and several windows were started in close succession to establish a federation. The federation was also run across the local TCP/IP network, running one federate on Windows 95 and another on Windows NT systems, and at this time we upgraded to RTI 1.3V6. The mode of operation of this case study is that, for each federate, the StartSimulation call is issued as soon as initialization completes. Simulation time starts advancing at this point.

Other studies were run on networks of Sun Sparc, HP-Unix, and X86 NT computers, to verify portability and data exchange. Since the MODSIM III language provides extreme portability, these compiled and ran without modifications to any application code.

In these experiments, the Rtiexec program was started first, then, one-by-one, the federates were started. The Rtiexec started a Fedex process, whose window opened 2-3 seconds after createFederationExecution was called. The calls on joinFederationExecution, repeated until a positive response, typically took 5-6 seconds. This appeared to create a 7-9 second period during which all the initial federates must be started.

Discrete-Event Simulations typically accept inputs at absolute simulation times. This assumption created problems with late joining federates, as might arise when the start of a federate is delayed beyond the initial 7-9 second starting period. It would be useful for the federation to not start until the essential members are present.

Peculiar behavior was encountered when a federate was abruptly terminated or crashed. Typically the other federates would hang up. This meant that the entire federation would have to be manually removed.

7 LESSONS LEARNED FROM JOB SHOP CASE STUDY

Lessons once learned become obvious, and are painful when made again. From the first case study we discovered some of these, and then found that we were not the first to learn them.

It has been proven that no protocol can exist such that two processes on a non-error-free network can each be aware of the precise state of the other. This is the "two-army problem" (Tanenbaum 1996, pp. 498-502), also described as the “coordinated attack problem” (Lynch 1996, pp 82-86).

Also, computer networks typically respond to overloads by dropping messages, and network protocols typically respond to messages that are lost by timing out and retrying. Due to this, most computer networks do not guarantee that separate messages will be received in the same order as they are sent (Tanenbaum 1996). Sequencing of messages is the province of higher-level protocols.

In practice, these limitations can be overcome, otherwise humanity would not be able to use ATM machines. Practical solutions such as the two-phase commit (Date 1986) are commonly used for this. However, these facts create considerable difficulty in initializing and starting a federation, for example, federates may not receive all object discovery messages before the attribute updates (Nielsen 1999). The best approach seems to be careful federate startup that may include a delay of perhaps 10 wall-clock seconds during federation startup.

It is unfortunate that an application must be so deeply concerned with details that should be handled by a communication protocol. The lesson that the first case study provides is that there is an opportunity for the programming tools to do a better job of providing repeatable initialization.

A related issue is that there is likely to be some difficulty with other features, such as object transfer, that require two federates to synchronize part of their state. The literature does describe this problem (Myjak 1999). His work describes some possible application workarounds but also proposes amending the architecture of the HLA to include additional services in this area.

The lesson from this is that the HLA will probably evolve, but the programming tools may be asked to provide practical solutions to problems that are impossible to solve in the theoretical general case.

The MODSIM III interface to the HLA was started with RTI 1.3V3. During the course of the development, the RTI was revised every few months, so we are currently demonstrating with RTI 1.3V6. These changes seemed evolutionary in nature, but they presumably affected performance or message traffic.

As other RTI versions are developed, the lesson here is that federates that pass tests on one version of the RTI may operate differently on other versions of the RTI. When network traffic is heavy, the differences will probably increase. Developers of large applications may need to be very aware of this.

A related lesson, sure to come back later, is that a small exercise may operate properly with the in-place local network or the internet, even while standard background traffic occurs. Large exercises may need to use dedicated networks or else they may malfunction, due solely to a different character of background traffic. This is unsettling to members of the Discrete-Event Simulation community because replicability is often expected and assumed.

Another lesson concerns difficulties with simultaneous events. Proper handling of simultaneity has always been important to people doing Discrete-Event Simulation. Simulated events access and modify shared data, so the way that ties are managed has global effects. CACI Products handle this by giving the simulation developer precise control. The SIMSCRIPT II.5 language offers the Simulation Modeler a rich set of tie breaking rules, and much the same tools are part of in MODSIM III. Developers of other Discrete-Event Simulation tools seem to find the need to provide similar capabilities. These difficulties are most recently described in (Wieland 1999).

Event times are often uncertain, especially when arrivals and service times are based on random distributions. It has been suggested that such a time value be represented as a range, with the simulation system free to handle events with overlapping time ranges as if they were concurrent events (Fujimoto 1999).

Object-oriented programming techniques can improve productivity by inserting common reusable code in base classes. However, design of base classes that are easy to use and understand is something of a black art. We like to measure reuse by counting lines of code or by counting keystrokes, then claiming that the smaller program is simpler and better.

The lesson that the Job Shop case study provides is that there are still some opportunities to improve code reuse.

8 CONCLUSIONS

What we learned from this research and development activity can be summarized in two parts:

1) Even though the HLA RTI 1.3 is a suitable data model for networked real-time training exercises, it falls short of addressing some key technical issues such as behavior modeling for networked Discrete-Event Simulation.
2) Use of the MODSIM III Discrete-Event Simulation language can save significant time and risk in building HLA-compliant simulation model federates.

These points help to set a direction for future work on tools to support networking of Discrete-Event Simulation models.

Further progress on our part will be posted on the following Web page: http://www.modsim.com.

REFERENCES

Booch, Grady, James Rumbaugh, and Ivar Jacobson. 1999. *The Unified Modeling Language User Guide*. Reading, MA: Addison-Wesley.

Dahmann, Judith S., Richard M. Fujimoto, and R.M. Weatherley. 1998. The DoD High Level Architecture: An Update. In *Proceedings of the 1998 Winter Simulation Conference*, 797-804. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Date, C. J. 1986. *An Introduction to Database Systems, Volume I Fourth Edition.* Reading, MA: Addison-Wesley.

Fujimoto, Richard M. 1999. Exploiting Temporal Uncertainty in Parallel and Distributed Simulations. *May 1999 Workshop on Parallel and distributed Simulations*.

Lynch, Nancy A. 1996. *Distributed Algorithms.* San Francisco, CA: Morgan Kaufmann Publishers, Inc.

Marti, Jed. 1999. *Object-Oriented Modeling and Simulation with MODSIM III.* La Jolla, CA: CACI Products Company.

Myjak, Michael D., Sean T. Sharp, Tom Lake, and Keith Briggs. 1999. Object Transfer In HLA. *SISO, Spring 1999,* presentation 140.

Nielson, Jeff. 1999. Federation Initialization and the RTI 1.3: Lessons learned from the JTLS-GCCS-NATO Federation. *SISO, Spring 1999,* presentation 039.

Page, Ernest H. 1998. The Rise of Web-Based Simulation: Implications for the High Level Architecture. *Proceedings of the 1998 Winter Simulation Conference*, 1663-1668. Institute of Electrical and Electronics Engineers, Piscataway, New Jersey.

Russell, Edward C. 1983. *Building Simulation Models with SIMSCRIPT II.5*. La Jolla, CA: CACI Products Company.

Tanenbaum, Andrew S. 1996. *Computer Networks Third Edition*. Upper Saddle River, NJ: Prentice-Hall.

Wieland, Frederick. 1999. The Threshold of Event Simultaneity. *SCS Transactions,* March 1999, Volume 16, Number 1, pp. 23-31.

AUTHOR BIOGRAPHY

GLEN JOHNSON is currently a Technical Fellow at CACI Products Company for the MODSIM III simulation language. He holds a BS degree in Mathematics and Computer Science from the University of California, Los Angeles.

Glen first joined CACI in 1967 to implement the SIMSCRIPT I.5 compiler on the major IBM computers of that era. He went on to create the SIMSCRIPT II.5 language compilers on leading mainframes and minicomputers. Glen designed the Process features of SIMSCRIPT and provided direct support for many major simulation efforts.

Glen has also worked at TeleSoft where he developed algorithms for the Ada language implementation to handle tasking, I/O, and memory management on large multiprocessor computers and produced the validated Ada 83 Compiler for the Intel 80960 military computer.

Glen also worked at Horizons Technology, Inc., where he was involved in interactive media, developed multimedia and video compression algorithms, and pioneered object-oriented techniques.

Since his return to CACI Products Company, Glen has continued work on programming languages and tools. His current interest is to make it easier for people to build simulation models that operate across computer networks and interact with the new generation of technologies.