

# Multi- Simulation Interface (MSI) for Complex Simulations

Aron Rubin  
Carl Hein  
Guru Prasad

Lockheed Martin Advanced Technology Laboratories  
3 Executive Campus  
Cherry Hill, NJ 08002  
{856 792- }9865, 9893, 9881  
{arubin, chein, gprasad}@atl.lmco.com

## Keywords:

High Level Architecture (HLA), Distributed Simulation, Distributed Interactive Simulation (DIS)

**ABSTRACT:** *This paper describes the Multi-Simulation Interface (MSI), a light weight and small foot print interface to network System of Systems (SoS) simulators. The emphasis is to have a low cost simulation model development environment to model complex systems. The paper discusses the differences between MSI and HLA, relative to increasingly complex and demanding distributed simulation environments.*

## 1. Introduction

Interconnecting simulations into common distributed simulation environments has shown to be an effective strategy for complex simulations. It enables simulations to specialize on different aspects or entities, while interacting within a larger context. It is often necessary to interconnect several simulators in a distributed simulation environment in order to leverage models, combine effects, distribute computations, and enable interactions of actual systems or detailed models of individual entities or platforms with each other. Starting in the 1980s several standard distributed simulation environments have been created.

In 2001 Lockheed Martin Advanced Technology Laboratories (ATL) was under contract to produce a discrete event simulation that could perform analysis on a virtual 50,000 node network at a faster-than-real-time rate executing on commercial-off-the-shelf (COTS) hardware. In addition, this network simulation (Figure 1) had to be distributed among nodes to enable virtual training use. The Defense Modeling and Simulation Office (DMSO) standard High Level Architecture (HLA) was selected and the development team created a conformant HLA interface with the RTI NG 1.3 software. However, as the simulation modeled tens of

thousands of network nodes, the HLA traffic required more bandwidth than a standard 10 Base-T network could handle. This observation has since been experimentally verified in a study that showed a performance degradation limit of 12,000 registered objects [1]. A lower overhead alternative to the DMSO

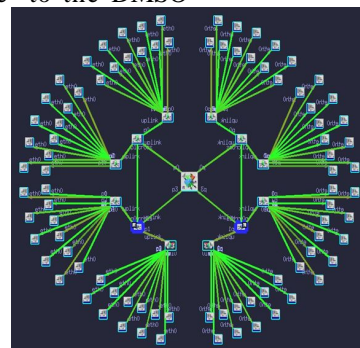


Figure 1. Network Simulation

distributed HLA had to be found with minimum impact on the project budget.

This paper describes the HLA alternate that resulted: the Multi-Simulation Interface (MSI)—a light-weight distributed simulation interface with a simple, intuitive interface. In its initial use for the network simulation the MSI showed an order of magnitude reduction in traffic. The MSI has been further expanded to address the issues of complexity, support, availability, and most

importantly, the cost associated with building complex, multi-domain systems. This paper also seeks to explain why the design of a new distributed simulation interface was necessary and how it relates to current and established standards in this field.

The MSI introduces some novel approaches to classical problems hindering the production of practical complex, multi-domain systems, such as social-economic, or engineering models. The problems like flexible interface interoperability, system organization preservation, and ad hoc federation are addressed through simple expression in the MSI interface. The solution to each of these areas, as well as some optimizations, is addressed later in this paper.

## **2. Background**

A fifteen year history of distributed simulation standards has led to the development of the MSI. Each new standard has built upon the features and lessons learned from its predecessors. Understanding the progression of design helps us choose meaningful remedies to current issues. Also we can realize the circumstance and gain perspective on why design decisions were made. This helps to separate designs that address needs and those that arose merely from circumstance.

### **2.1 Project Specific Point-to-Point**

Initially, and to a large extent still to this day, simulators were interconnected point-to-point with ad hoc interfaces. This meant that a new interface was designed for each pairing of simulators. As these interfaces grow and more simulators are combined, scalability issues need to be revisited for each new design. Software becomes obsolete or unmaintainable in the absence of the use of any standards due to the costs of continuing stakeholder support.

### **2.2 SIMNET and DIS (Distributed Interactive Simulation)**

In the early 1980s, the ARPA SIMNET program focused on connecting real-time tank training simulators through conformance to interoperable standards. It provided a communications framework that later evolved

into the Distributed Interactive Simulation (DIS) run by STRICOM. DIS standardized broadcast message formats to disseminate state and event information. Each simulation must process every item placed on the network. The amount of CPU overhead and network bandwidth required to conduct large simulations quickly becomes excessive [2].

DIS was intended for loosely synchronized training simulators over high-speed local area networks (LANs). The simulations and other DIS connected entities were considered to be independently synchronized to real-time. The situation did not require precise coordination between the small number of simulated entities and was relatively insensitive to message loss or delays. The simulation of tasks requiring precise coordination on short time scales is generally considered beyond DIS capabilities.

### **2.3 ADS and ALSP (Aggregate Level Simulation Protocol)**

By the early 1990s, it became clear that DIS had limited applicability in increasingly complex and demanding distributed simulation environments. Needs arose to connect simulators not linked to real-time and for uses beyond training. ARPA set out to develop a new framework in the Advanced Distributed Simulation (ADS) project. This new framework became known as the Aggregate Level Simulation Protocol (ALSP). ALSP emphasized three main features over DIS—central logical clock management, common data representation system, and architecture independence [3].

### **2.4 HLA (High Level Architecture)**

HLA was developed by the Defense Modeling and Simulation Office (DMSO) to support reuse and interoperability across large numbers of different kinds of simulators [4]. Designed to support and supplant both DIS and ALSP [3], HLA contains sophisticated technologies beyond DIS that are encapsulated in a reusable Run Time Interface (RTI). The HLA Baseline Definition was completed in 1996, was adopted by the Object Management Group (OMG) in 1998, and approved by the Institute of Electrical and Electronic Engineers (IEEE) as IEEE Standard 1516 in 2000.

HLA contains the most sophisticated features of any current simulation interconnection specification. Certainly in its IEEE 1516 incarnation HLA provides support for the broadest mix of time and data management situations. HLA employs a CORBA inspired publish and subscribe data management scheme which allows ad hoc discovery of simulation entities and spatial management to distribute state and event information where needed. HLA supports heterogeneous mixing of causally constrained and unconstrained federates. The specification of delivery mechanism on an per-object-type basis enables HLA to also support heterogeneous data updates.

Until 2000, DMSO contracted the building of a reference implementation of HLA. This reference implementation was released as a binary-only distribution for a number of specific distributions. Because of this limitation in supply, the platforms for which HLA was available quickly became obsolete. Several commercial vendors implementing HLA have since started regular platform releases, but the cost often prohibits casual use.

#### 2.4 MSI (Multi- Simulation Interface)

Issues of performance, complexity, support, availability, and cost arose with HLA. Even though the HLA standard was well designed, there are still gaps in its usability. The Multi-Simulator Interface (MSI) was created to address some of these issues. The MSI started as a lightweight, HLA-like engine that communicates using XML encapsulations [5]. It connects (Figure 2) simulations by maintaining the appropriate levels of coherency in shared state data and by preserving causal relationships. The MSI was intended to provide a parallel design that is readily usable and performs well in the most common use cases. If HLA had been designed after XML reached its current popularity, HLA may have leveraged XML more like MSI does. In addition to supporting the salient features of HLA, MSI was tailored around much Faster-Than-Real-Time (FTRT) simulation and designed specifically to support Systems-of-Systems (SoS) simulations.

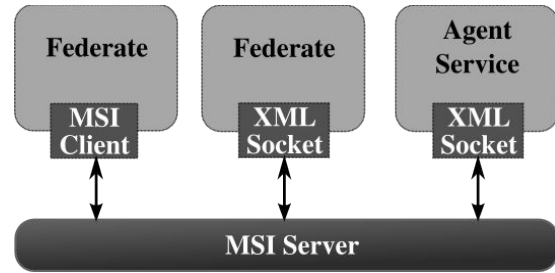


Figure 2. MSI Federation

### 3. Interoperability

#### 3.1 Interoperable and Portable Architecture

Traditionally, a system was minimally interoperable if it could connect to a fixed set of other systems. More recently, interoperability implies interface standards available to any system conforming. The MSI uses an XML stream through a direct socket connection for communications. This enables the MSI to be used from any programming language that can use sockets (C, C++, Java, Ada, Lisp, Perl, etc.). Also, the MSI was written with no library dependencies and cross-platform code that make it portable to all the major OS platforms (Linux, Solaris, Mac OS X, Microsoft Windows, IRIX, HPUX, etc.). The MSI is a single executable file and is distributed with example code for the simulator/federate side interface.

#### 3.2 Native Data Representation

Many frameworks and middleware software force all clients to produce neutral data representations. Each data receiver must be written to parse the same neutral data representations. To the degree that the actual content of the data is present, all clients then speak the same language. Sufficient metadata is added to each envelope of data that any system could then parse the semantics of the envelope, even though it may be incomplete for that receiver's representations.

However, there is a high cost for representation neutrality in terms of processing and bandwidth. Processing costs occur every time data must be transformed from one form to another. More importantly, representation neutrality forces all transmissions to be verbose. It would be optimal if there was a way to gain the flexibility of a representation agnostic form without sacrificing

performance.

The MSI uses a Rosetta Stone (Figure 3) approach to translation and data interoperability to achieve optimal results. Instead of transforming the data on input, MSI was designed to leave data in its original form for efficient transport, and then the data receiver (or MSI client-side library) can transform the data into the best local form. If the data receiver can parse the incoming form no processing cost is incurred. The only requirement for this methodology to work is that the data be, at a minimum, transformable. XML provides an encoding that, when well formed, does not require a context sensitive parser and is easily transformable. Indeed there exists a standard language, XSLT, specifically designed to specify arbitrary XML transforms.



**Figure 3. Common Data Translation**

The XSLT needed to transform data produced by a data sender to and from a neutral representation acts as a face of the Rosetta Stone. This XSLT can be provided or referenced once as an object is published and then applied as needed thereafter by a subscriber. If a subscriber does not process the neutral form or as an optimization, it may compile the incoming XSLT to transform a foreign format directly to its own. This completes the Rosetta Stone analogy if the different XSLTs describing the same data are thought of as being different faces of the Rosetta Stone. This concept is known as an inter-ontology mapping.

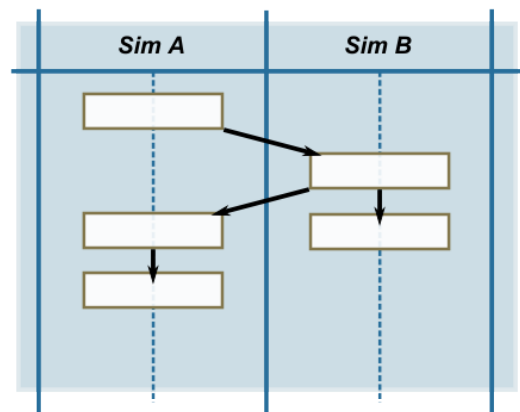
#### 4. Causality

The MSI is designed to be as lean as possible while still providing sufficient functionality. Lean design means that the intrinsic needs of a simulation interconnection engine are examined

to determine its core behavior. Focusing on these needs contributes to the reduction of the complexity of the design and interface.

#### 4.1 Causal Hazards

In most applications, one must be cautious when designing parallel software. Parallelized software often employs synchronization mechanisms to avoid various hazards that can occur. These are classically data hazards such as read-before-write, write-before-read, and control hazards such as dead-lock and live-lock. Discrete event simulation (DES) applications add another complication: the maintenance of causality. Events are ordered not just by data and control flow constraints but by temporal constraints enforced on time tags. For example, if a certain control flow constraint specifies that events must occur *A* then *B* then *C* (Figure 4), and a system executes events  $a_1$  then  $b_0$  then  $c_0$ , it may satisfy the looser constraint of control flow ordering and still violate causality constraints. Many communication mechanisms exist to synchronize data under such conditions; however, in addition to data synchronization, one of the key roles of a simulation interconnection engine is to maintain the causality of events. That is to ensure that no event may have occurred in one application that may have occurred differently (or not at all) based upon events in another application. The definitive description of this concept is reported by [6], which states events *a* and *b* are concurrent or independent if neither can causally affect each other [6].



**Figure 4. Causal Relationships**

Many systems do not use mechanisms to maintain causality and rely on independent, real-time clock synchronization to approximate proper ordering. Two branches of methods exist to handle causal hazards. Conservative methods do not allow violations of local causality constraints. Messages are passed which keep simulations in lock step but allow for parallel operation where possible. Optimistic methods are more loosely synchronized where one simulation may execute beyond the conservative time boundary if its state could be returned upon the determination of a violation. Although optimistic time management has the potential to execute faster than conservative if the scenario provides a low probability of violation and the computational cost of state restoration can be mitigated [7], these conditions may not be observed in practice.

Though HLA can support both conservative and optimistic time management, the support of optimistic simulation comes with high development cost. Any simulation that participates in the optimistic scheme must be written such that its complex state can be stored and re-stored. This may be tedious or impossible depending on the content of a simulation. A simulation may need to bypass the compiler's management of the program stack to support state restoration, and to journal every bit of memory that can be causally affected. Likewise any simulation interconnection engine that supports optimistic time management must journal its data distribution so simulations discovered to be violating causality could ensure that all resulting retractions reach the original recipients. In order to remain light-weight, the MSI does not support optimistic time management.

#### **4.2 The Lean Time Management System**

The MSI time management system does not force explicit specification of many behaviors; it will exhibit different behaviors based on usage and cover most situations of time management. The MSI time management system can, for example, mix temporally unconstrained with time-constrained and time-constraining simulations. A great range of causal strictness can also be expressed through particular use of the same time management command. However, the MSI presently has no mechanisms to

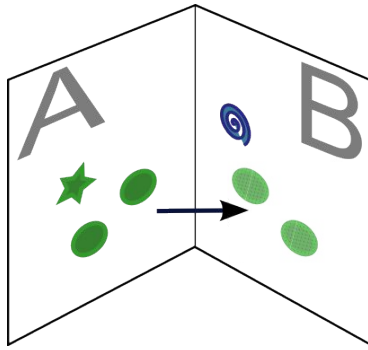
facilitate optimistic time management.

The system is based on the MSI server maintaining a Global Virtual Time (GVT), which is advanced as it is causally safe to do so. Each constrained simulation reports the time to which it can be advanced without violating its local causality constraint, called the Time Of Next Event (TONE). This typically matches the time of the next event that will occur in that federate, but it may be artificially inflated to cause loose synchronization (less overhead but less guarantee of accuracy). After a barrier synchronization on the reporting of each federate the next value of the GVT is determined from the lowest time reported. The federates are then sent a message granting them permission to advance to the GVT. If a simulation/federate receives a message or update after reporting its next event time, it must report a new value for its TONE. It is possible for this technique to lead to many messages for federations where the number time advancements outweigh the computation performed per message. However, such a federation would have limited utility and is therefore unlikely to occur, which indicates the TONE-based method will tend to be viable.

### **5. Data Management**

#### **5.1 State Data Coherency**

Frequently simulations will need to access the state of models or other data stored in another simulation where the production of that data took place. In addition to ensuring causal relationships between simulations, another key role of a simulation interconnection engine is to provide mechanisms to maintain the coherency of this data. Specifically the engine may control the degree of accuracy with which any copy of state data matches the original (Figure 5). This is achieved by facilitating or orchestrating the transfer of data between federates. This capability interacts with causal and data discovery mechanisms so tightly as to make decoupling impractical. Some federations or specific federates may not be concerned with coherency but it must be supported at a minimum.



**Figure 5. Mirrored Data**

Several different methods keep data coherent, but the most important aspects are timing and distribution. To maintain proper coherence in time, a simulation interconnection engine must ensure that data produced at a certain logical time in one federate is reflected in other federates as they pass through the same logical time. The actual contents of transmission that synchronize state aside, the delivery of a message must occur before another federate in the causal chain advances its logical clock past the logical message generation time.

## 5.2 The Lean Data Distribution System

The MSI implements a publish and subscribe data broker. The MSI presently does not check the data format for validity; therefore, it does not require a separate data format specification (like the HLA Federation Object Model). The communications are expected to be pre-validated using one of the many XML validators available (most web browsers). This greatly reduces MSI's setup time, message processing overhead, and learning curve. Also, not being locked to a predetermined data format allows for dynamic data types.

The MSI has a flexible publish and subscribe system. Five commands are associated with the MSI data broker: (1) publish, (2) subscribe, (3) update, (4) unsubscribe, and (5) destroy. Simulations/federates may subscribe to object names in addition to object types. This allows simulations to subscribe to specific objects of a type without needing to receive updates of all objects of that type. The update command is both an incoming and outgoing command. When a simulation/federate receives an update command, it is expected to reflect the new values of that object [8]. In addition, a federate

may specify particular attributes of an object or object type. For example, if an object has attributes 'name', 'x', 'y', and 'z', a federate that only considers two dimensions may choose to subscribe only to 'name', 'x', and 'y'.

## 5.3 Caching

With the need for ad hoc simulation federations, either to support dynamic scenarios or web services, the impact of a federate *join* and *exit* should be minimized. Even with a classical simulation scenario, a simulation interconnection engine would either need to prevent object updates or save all object updates until all federates have joined. To better support this type of operation, the MSI has the ability to cache object updates.

Under one use case, various federates may join, participate in the simulation for a short time, and then exit leaving data behind. The federates may be testing a hypothesis in simulation or executing a monte carlo. If some state was not cached, the data would have to be pulled anew from its original sources.

## 6. Managing Complexity through Organization

This section will discuss some of the specific techniques that are available to manage complex behavior in MSI. The techniques could be seen as optimizations; however, unlike many optimizations, they do not require unintuitive redesign but rather organization and tagging.

### 6.1 Hierarchy and Systems of Systems

One technique to manage complexity is recognizing and preserving the structure of a system's organization. MSI data distribution system is aware of hierarchy if specified. As objects are published or updated, the relationship to a super-object may be specified through the use of the "parent" attribute. This has particular effects with respect to whether subscribers to that object or its parent object will receive updates. Setting an object, *A*, as the parent of another object, *B* means that: (1) a federate subscribed to *B* will receive updates for *B*, and (2) a federate subscribed to *A* will receive updates for *A* and *B*. Table 1 provides an

example of this behavior.

**Table 1. Example of Systems of Systems Mappings**

Publishings		
Client 1	publishes entity state subsystem	"stryker0.state" child of "stryker0"
Client 2	publishes communication subsystem	"stryker0.comms" child of "stryker0"
Client 2	publishes messaging subsystem	"stryker0.comms.traffic"
Client 3	publishes radio subsystem	"stryker0.comms.radio" child of "stryker0.comms"
Client 4	publishes weapons/fire subsystem	"stryker0.fire" child of "stryker0"
Subscribes		
Client 1	subscribes	stryker0.fire
Client 2	subscribes	stryker0
Client 3	subscribes	stryker0.comms
Client 4	subscribes	stryker0.state, stryker0.comms.traffic
Updates		
Client 1	receives...	stryker0.fire
Client 2	receives...	stryker0.state, stryker0.comms.radio, stryker0.fire
Client 3	receives...	stryker0.comms, stryker0.comms.traffic
Client 4	receives...	stryker0.state, stryker0.comms.traffic

## 6.2 Dynamic Context Culling

Among the optimization techniques facilitated by the MSI is the ability to cull unnecessary traffic. In contrast to filtering, which conditionally blocks specific traffic at the recipient, traffic culling refers to the conditional removal of specific traffic. HLA uses a method called a routing space to specify culling conditions performed in the RTI (HLA server software). Routing spaces are linked with data structure declaration in the HLA federation object model (FOM). With MSI, culling decisions can be performed at the client or, in a more limited form, in the server. When using the MSI, culling conditions are expressed as subscription qualifiers. A federate may reduce the traffic that it has requested, for example, by subscribing to named objects instead of types where appropriate.

It is more efficient to subscribe restrictively first and then more broadly as needed. This is the foundation of MSI's powerful dynamic-context-culling capability for the client side. It is simple, but it gives a lot of control to a user and the ability to make decisions based on non-scalar data values. The client-side dynamic-context-culling algorithm is as follows:

- Subscribe to specific attributes to determine context
- Calculate distance to context
- If close then subscribe to other attributes.

Example tracking visibility with object 'C' at  $x=0$   $y=0$   $z=0$ :

- Send `<subscribe type="X" attributes="x,y,z">`
- Receive `<update id="0" type="X" name="A" x="9.1" y="0" z="31.4"/>`
- Receive `<update id="1" type="X" name="B" x="2" y="0" z="1"/>`
- Calculated  $|C-A| = 32.7$   $|C-B| = 2.23$
- $|C-B| < 5$  so send `<subscribe name="B" attributes="shape,color"/>`

If values are changing rapidly, there is cost to each federates receiving updates, however small, prior to culling. This is why the MSI also offers a limited server-side dynamic-context-culling. A user can qualify a subscription with conditions that are evaluated on value state changes. The values are then assumed to be scalar. With the combination of MSIs client and server side culling, a user can ensure that the only updates that are received were necessary for the calculations being performed locally.

## 7. Conclusions

The MSI is a simple but powerful tool to connect simulations. The MSI builds on the efforts and designs of preexisting standards as well as integrating new, established industry standards. Design and architecture of the MSI were guided by concerns for usability, efficiency and lean design. The MSI requires a minimal learning curve. Even though the MSI is not thought of as a replacement for HLA, it may serve as a boot strap or an alternative when HLA exceeds the budget or complexity of a project. The MSI is designed to be lean while still providing

sufficient functionality. The approach provides power through the combination of a minimal number of simple intuitive interface elements.

New uses of distributed simulation environments have been enabled by technologies incorporated into the MSI. The MSI's caching ability and data distribution management allow for ad hoc distributed simulations. Agent-based and service oriented interaction are facilitated by MSI's approach to data representation transformation.

## 8. References

- [1] Hong, Su-Youn, Kim, Jae-Hyun, and Kim, Tag Gon (2005). Measurement of RTI Performance for Tuning Parameters to Improve Federation Performance in Real-time War Game Simulation. Proceedings of the 2005 Summer Computer Simulation Conference.
- [2] Hofer, Ronald C. and Loper Margaret L. (1995). DIS Today. Proceedings of the IEEE, Vol. 83, No. 8.
- [3] Weatherly, Richard M., Wilson Annette L., Canova, Bradford S., Page, Ernest H., and Zabek, Anita A. (1996). Advanced Distributed Simulation through the Aggregate Level Simulation Protocol. Proceedings of the 29th Annual Hawaii International Conference on System Sciences.
- [4] Defense Modeling and Simulation Office (1998) High Level Architecture Interface Specification, Version 1.3: Washington D.C.
- [5] Interface Spec 2002, <http://msi.sourceforge.net/>
- [6] Lamport, Leslie (1978). Time, Clocks, and the Ordering of Events in a Distributed System. Communications of the ACM, Vol. 21, No. 7.
- [7] Fujimoto, R.M. (2001). Parallel and Distributed Simulation Systems. Proceedings of the 2001 Winter Simulation Conference.
- [8] MSI Brochure 2003, <http://msi.sourceforge.net/MSI>

## Author Biographies

**ARON RUBIN** is an engineer with Lockheed Martin Advanced Technology Laboratories (ATL). His primary area of applied research has been in modeling, simulation, and development of new, emerging and emergent network architectures. He received his BS in Computer Science from the engineering school of Lehigh University. Over his nine years with ATL, Mr. Rubin has worked on a wide range of projects including performance modeling, information assurance, multi-dimensional scheduling, distributed simulation, and information systems management.

**CARL HEIN** has 20 years of experience in modeling complex systems at multiple abstraction levels at Lockheed Martin Advanced Technology Laboratories (ATL). He focuses on algorithmic and behavioral modeling, performance modeling, and resource modeling to facilitate complex systems analysis and distributed software development.

**DR. GURU PRASAD's** research interests are in simulation, architectures, and model integration for Complex Systems. He has over 15 years of experience in modeling and simulation and related technologies, including constructive-virtual and physics based behavioral modeling, visualization and simulation architectures, distributed and parallel computing, DIS, and HLA. He was involved with the development of several types of simulators and DIS standards. He has over three years of experience with spacecraft command and control technologies, satellite mechanism, docking system, and simulations. He has taught advanced simulation courses at the Simulation Institute in Central Florida. He has a doctoral degree from University of Central Florida in Industrial Engineering – Simulation Systems.