HLA Federation Performance: What Really Matters?

Ben Watrous
Len Granowetter
Douglas Wood
MAK Technologies
68 Moulton St.
Cambridge MA 02138
(617) 876-8085
bwatrous@mak.com, lgranowetter@mak.com, dwood@mak.com

ABSTRACT: *HLA Federation developers often struggle to achieve their performance goals and must tweak their chosen RTI to meet those needs. RTI implementations offer myriad configuration options to control the behavior of the RTI and take advantage of vendor specific features. Eventually, users find configurations that meet their needs in a particular federation with a particular RTI implementation. However, without a deep understanding of how various factors affect the performance of an HLA federation, the process of optimizing a federation often involves a lot of trial and error. This paper tries to answer the question of "What really matters" when trying to optimize the performance of an HLA federation.*

We will look at four important sets of performance constraints: 1) Requirements levied by the HLA Specification on RTI implementations; 2) Choices made by various RTI implementations; 3) Demands of each federation (services used, number of objects, etc.); and 4) Limitations imposed by physical resources available to federates and the RTI. We will examine some of the "conventional wisdom" surrounding HLA performance, and discuss which rules of thumb are appropriate across all HLA federations, which apply only to certain RTI implementations or federation designs, and which are outdated entirely. For example, although there is a common assumption that the use of Time Management in a federation adds a lot of overhead to all federates, it is quite possible to implement time management in such a way that it imposes almost no performance penalty at all on non-time-managed federates. On the other hand, some HLA requirements that appear quite innocuous, such as the guarantee of unique names and handles, have a surprisingly significant impact on federation performance.

1. Introduction

The performance of an HLA federation depends on many factors. Design of the Federation Object Model, choice of RTI implementation, distribution of simulation responsibilities across the federates, and configuration of the network and RTI can each have a significant effect on the success or failure of the federation. When making these decisions, it is important to understand the various performance drivers, and how they combine to define overall federation performance.

Performance drivers fall into four broad categories:

First, the HLA Standard itself levies many requirements on all RTI implementations. Therefore, there will be many similarities in how different services are implemented across RTIs. For example, Time Management requires coordination among the time managed federates' Local RTI Components (LRCs).

On the other hand, HLA was designed to allow implementors flexibility in their choice of algorithms and supported media while allowing federate developers the ability to select the implementation that best meets their needs at a given time. So the second category of performance drivers are the techniques employed by an RTI, that may enhance one federation strategy while degrading another.

The third set of performance drivers are the demands placed on the RTI by the federation itself. This includes the scale of the federation (number of federates, number of objects, amount of data that needs to be exchanged among the federates, frequency of updates and interactions), the set of HLA services that the federation requires, and the ways in which the federates agree to interact with the RTI. For example, some federations require an initialization stage, where all joining, subscribing, publishing, and registration takes place.

Finally, the physical resources upon which a federate operates can affect performance. This includes the amount of bandwidth available, speed of networks, processing power, memory, and number of machines available.

When optimizing a federation for performance, it is important for federation designers to understand which

performance drivers are the current bottlenecks. This enables them to recognize when there is an underlying physical constraint that needs to be relaxed, when an alternative RTI or configuration can offer better results, and when they must redesign the federation in order to achieve their goals.

Users often make generalizations about HLA itself based on specific RTI implementations, or experience with specific federations. However, deficiencies in one RTI may not exist in another RTI or even in a different version of the same RTI, and lessons learned from one federation may not be applicable to another.

Based on our experiences as both a federate developer and an RTI implementor, we attempt to shed some light on the impact of each of the performance drivers. We will examine some of the "conventional wisdom" surrounding HLA performance, and discuss which rules of thumb are appropriate across all HLA federations, which apply only to certain RTI implementations or federation designs, and which are outdated entirely.

2. Impact of Specific HLA Requirements

Although the HLA specification has been designed to allow great flexibility to implementors, the specification often lends itself more naturally to one implementation over others. Some of these choices are obvious due to the target environment or the natural flow of federate life cycle. Others are more subtle and based on the contract offered by individual services. RTI implementations distinguish themselves through their implementation choices for each HLA service. But it is important to recognize the areas that offer less flexibility to the implementors since these are the areas that can be described as limiting the performance of HLA itself.

For example, while it may be possible to design a fully distributed and still compliant RTI, services such as Synchronization Points are much more naturally achieved using a central controller. HLA's synchronization points make a very simple synchronization guarantee - that all interested federates have received the Announce Synchronization Point callback and responded to it before any federate receives the Synchronization Point Achieved callback. Even this limited form of synchronization naturally leads to a centralized algorithm. While it is possible to achieve in a fully distributed network, it would require many more messages to be exchanged among LRCs. Synchronization points are not often a limiting factor in federation performance, but this is just one example of how the HLA Specification naturally leads to a particular implementation choice.

In some cases, it appears that there are many different ways to implement a particular HLA service group. However, there are often some small, innocuous-looking requirements ("shall" statements in the Specification), that effectively preclude a choice that would otherwise be more efficient. Sometimes, the cost of one of those "shall" statements is not obvious.

Conversely, there are some aspects of the specification that are commonly believed to add large amounts of overhead to RTI implementations, but actually do not. For example, time management is commonly believed to add a large amount of overhead to HLA. However, the standard was carefully designed to allow implementations that minimize the impact of time management on federates which do not use time management services.

Listed below are a few of the key areas where the specification does directly impact performance. The constraints may allow several implementations, but lead most naturally to only one or a few designs. Some of these constraints are imposed by more than one aspect of the HLA specification, for example there are many reasons (such as the Destroy Federation Execution service call and the MOM federation object) that the RTI must maintain a list of joined federates available in some form to all LRCs. In the interests of clarity, each operation is treated independently.

2.1 Create Federation Execution / Destroy Federation Execution :

Even simple services like the Create Federation Execution and Delete Federation Execution service calls can place certain constraints on the RTI implementation. Create requires that the RTI guarantee that the federation name is unique. That implies that the local federate must have a way of querying for existing federations. This could be done in a fully distributed fashion by querying each existing LRC, but it is certainly much more efficient and straightforward if the RTI maintains a central list of federations. Similarly, the Destroy operation requires an RTI to provide the LRC with a method of querying for joined federates. There are many ways of achieving this goal, but the obvious choice is to provide a central location that tracks joined federates.

These requirements naturally lend themselves to a centralized solution and make a fully distributed RTI implementation much more difficult. If the LRC were not required to throw the FederationAlreadyExists exception, the RTI could optimize the CreateFederation execution. In that case, the RTI might simply make the assumption that either the federation name is unused or that, if the name is in use, it is safe for federates to join the existing federation. Similarly, the requirement for the RTI to

throw the "FederatesCurrentlyJoined" exception on Destroy levies the requirement that each LRC have access to the list of joined federates.

Without the requirement to throw the FederationAlreadyExists exception, the RTI could do far less work in CreateFederation execution since the federate might be able to avoid a distributed operation to check and update the list of federations. Several existing implementations offer a "lightweight" mode in which this is possible and greatly speeds up federation creation. However, it leads to unexpected and possibly confusing results if multiple independent federations attempt to use the same name simultaneously.

The CreateFederation service is just one example, the requirement to throw detailed exceptions in the case of federate errors has performance impacts throughout the specification. Implementing proper exceptions for all the error conditions is a large burden on RTI implementors, but it benefits the users greatly and RTIs are free to offer the ability to lower overhead via configuration options for users who can afford to relax the restrictions.

2.2 Publish/Subscribe/Advisories:

The most time consuming startup operation for most federations is the publication and subscription step. The cost is almost entirely due to the requirement for the RTI to generate advisories when subscribers are interested in published data.

Advisories were designed to allow publishers to optimize the data they send by updating only attributes and interactions that have potential consumers. However, in real federations this benefit rarely applies – federates rarely send updates that no other federate in the federation cares about, otherwise they would simply not publish the data. So, while advisories may save some work at a small number of federates, it has never offered the intended performance benefits.

At the same time, providing accurate advisories is a very expensive operation for the RTI itself. The major cost of advisories is the requirement that all federates have access to the current subscription interests of all other federates down to the attribute instance level. Most of this cost can be moved to the federation startup phase with careful federation design, but, as shown below, the heaviest cost is incurred by late joining federates. Because of this, advisories are often considered to offer the lowest performance-cost ratio of the HLA specification and most RTIs offer a means to disable them [3].

However, the true cost of advisories is heavily dependent on the underlying RTI implementation. For example, an RTI might choose to maintain the subscription database at a centralized location. In this case, only the central location must have full knowledge of all federates' subscriptions and provide advisories indirectly to the joined federates. In another RTI implementation the same subscription knowledge may be required to support other algorithms. For example, federates may need to maintain subscription databases in order to support sender-side filtering of updates and interactions the benefits of which may greatly out weigh the cost of maintaining the largely static subscription database.

So, while the specification levies the burden of maintaining the subscription database on all RTIs for what is often a small benefit, the cost to the federation can be mitigated by the RTI implementation.

2.3 Late Joiners:

HLA requires RTIs to allow federates to join at any time between federation creation and destruction (excluding save or restore in progress conditions). Federates which join after the initial startup phase of a federation are generally referred to as "late joiners." Whenever a federate joins a federation it must be given a partial view of the current state of the federation. Since the existing joined federates were already made aware of (at least some of) this state, support for late joiners requires retransmission of data.

The concept of late joiners applies to all classes of distributed architecture that allow dynamic membership, since new comers will always need to be informed of the current state of the system. However, the publish and subscribe based architecture along with the goal of minimizing retransmission of data present particular challenges to the RTI implementation. For example, a well behaved DIS federate will periodically heartbeat the state all of its objects so a late joiner will gradually build up the current state of all objects in the system with no special handling. This is certainly not the case in the RTI.

First, late joiners add to the cost of issuing advisories. The RTI must provide the federate with the current Advisory state as the federate issues its publications. Since the late-joining federate is likely to have missed the subscriptions of other joined federates, the LRC cannot simply gather subscription information as other federates subscribe. In an RTI which maintains the subscription database at each LRC this may require a retransmission of the entire subscription set from each joined federate. If the federate joins during a running exercise, this flood of subscription information may negatively impact federation performance as a whole. An RTI may reduce the impact by generating advisories at a central component, but only at the cost of other optimizations possible when the LRC

has local access to the subscription database.

There is an even larger cost to accepting late joiners. Once the late joiner has issued its own subscription interests, the RTI must inform it of all registered attribute instances which match its subscription interests for discovery, and the federate is likely to respond by issuing a request attribute value update for every attribute instance it discovers. The currently joined federates will then retransmit the state of all requested attributes even though they may not have changed since the last update.

HLA 1516 and HLA Evolved offer an Auto-Provide Switch in the FOM which guarantees that whenever an attribute comes into scope for a subscribed federate the owning federate will receive an update request for that attribute. This eliminates the need for the late joiner to transmit a request for each discovered attribute instance, but the burden is simply moved down to the RTI level.

The costs associated with late joiners may vary widely among RTI implementations, but there is no way for an RTI to entirely avoid the cost without making trade-offs elsewhere (such as DIS-style heartbeats). This means that all HLA federations should handle late joiners carefully.

Indeed, even federations that only allow federates to join during specified startup phases may be affected by HLA's handling of late joiners. In the case of object discovery and retransmission of initial attribute state - not only must all federates be joined, they must also be subscribed before the object registration in order to discover the object when it is first registered. One strategy is to break the startup phase down further into a publication stage followed by a subscription stage followed by a registration stage. But such a strategy essentially ensures that the slowest federate will dictate startup time and, as mentioned in the discussion of synchronization points, federation synchronization is an expensive process in itself.

2.4 Updates and Interactions

HLA is heavily optimized for the sending of updates and interactions. This is unsurprising, given that the bulk of the communication between federates takes the form of updates and interactions. The specification was designed to minimize the number of layers between the federate and the underlying media and to limit retransmissions by allowing federates to update only attributes whose values have changed.

Still, there are several areas that affect performance in ways that users often don't anticipate.

For example, HLA requires an RTI to deliver a set reflected attributes in what is known as a passel. The HLA 1516 specification provides the following definition of a passel:

"A group of attribute handle/value pairs from an *Update Attribute Values* service invocation that are delivered together via a *Reflect Attribute Values* . service invocation. All pairs within the passel have the same user-supplied tag, sent message order type, transportation type, receive message order type,time stamp (if present), and set of sent region designators (if present) [1]."

In other words, the RTI is required to divide the set of attribute handle, value pairs passed to the Update Attribute Values call into sets of attributes with identical parameters. This process is known as passelization. Each passel is passed to the subscribed federate in a separate Reflect Attribute Values invocation.

Passelization occurs whenever a single attribute value update is made with attributes having different parameters. A simple example is a federate updating a set of owned attributes some of which are associated with the Best-Effort transport type and some of which are associated with the Reliable transport type. Federate developers commonly expect the data from such an update to be delivered to subscribed federates in a single reflect. But, due to the passelization requirement, the RTI must deliver the attribute values in two separate reflects.

Passelization is a costly operation. The RTI may choose to implement passelization in a number of different ways, but each of them is likely to incur more overhead than an update of attributes belonging to a single passel. In the example above, the RTI may choose to break the attributes up on the sender side and send the best-effort attributes via its best-effort channel and the reliable attributes via its reliable channel. This will likely result in a copy of the attribute values and an additional network operation. Instead, the RTI may decide to send all the attributes via its reliable channel and perform the passelization on the receiving end, but a reliable transmission is generally more expensive than best-effort. Either way passelization adds overhead to attribute transmission, and this overhead only increases when DDM comes into play.

Federation designers should be aware of issues such as passelization and the strategies used by their RTI implementation as well as any optimizations offered. For example, federates may be able to reduce the cost of passelization by submitting attribute value updates pre-packaged as passels and some RTIs may offer configuration options which eliminate passelization all together (at the cost of compliance with the specification).

2.5 Ownership Transfer

Transfer of ownership is a coarse grained mechanism in HLA. The HLA ownership semantics preclude

instantaneous or even entirely predictable transfer rates when new federates are added to the federation. This is because HLA's ownership transfer mechanisms require messages to be passed between federates. Since federates involved in the transfer must wait for responses from other federates (which may be occupied performing other processing), ownership transfer should be treated as a heavy-weight event by federates.

Federates may be tempted to use ownership transfer to simulate remote method invocation on shared objects. But HLA enforces exclusive attribute ownership semantics rather than sharing semantics and its ownership transfer mechanism is not well suited to algorithms requiring rapidly changing ownership. Instead, federates should use other mechanisms such as interactions to achieve the same goals.

2.6 Time Management

A common misconception is that time management imposes a tremendous burden on the RTI even when not in use. It is true that there is likely to be some additional startup cost and memory usage to support time management; however, it is quite possible to design an RTI in such a way that non-regulating, non-constrained federates (even in a time managed federation) pay almost no penalty during federation execution.

For time managed federates, the story is entirely different. In that case, time management has a heavy impact on both the RTI implementation and federate performance.

Since time management is a synchronization algorithm like the synchronization points described earlier, it requires communication between all time managed federates. Any distributed algorithm which requires contributions from all nodes will be relatively expensive. Even if all federates respond as quickly as possible to all time state changes, the rate of time advancement will be limited by the number of messages passed within the RTI to achieve each advance. This makes time management a common target of benchmarks and an RTI implementation differentiator.

Time management also enforces similarity between RTI implementations. Because time advancement requires contributions from all time-regulating federates, it is generally far more expensive to implement without a central time manager to coordinate the individual LRCs. Similarly, the requirement to deliver time-stamped messages in both time-stamp order and sent order for messages from a single federate poses severe restrictions on the RTI. To guarantee time-stamp order, federates must queue time-stamped messages. Queuing messages is always an expensive operation since it implicitly requires storage of the message for delayed delivery. In order delivery with respect to the sender can be even more difficult to guarantee given that most implementations send best-effort messages via UDP and UDP makes no guarantees regarding message ordering. This means that RTIs must implement a message ordering algorithm in order to satisfy the ordered delivery requirement or simply gamble on the idea that out of order messages are rare. There are many other time-management related issues (for example the fact that the federation management operations are not time-managed leads to some very tricky requirements), all of which impose restrictions on the choices of time management implementors.

2.7 Data Distribution Management

Data Distribution Management allows a federate to refine its interests at the attribute instance level in order to reduce both the transmission and reception of irrelevant data [1], [5]. These obvious benefits must be weighed against the added complexity of using the DDM services, and underlying implementation factors impact its effectiveness.

First, HLA users often expect to be able to use DDM to reduce network bandwidth . However, most DDM implementations employ UDP multicast (for best effort) to route data between federates [2], [3]. While multicast will reduce the message processing at the receiving side (by both the RTI and the federate), the messages are still sent to the network. So, no actual reduction of network bandwidth occurs.

Secondly, any DDM implementation that fully complies with the interface specification requires the exchange of the region information, specifically to perform "perfect filtering" and to support advisories [4]. But, "perfect filtering" in the DDM sense is a misnomer. The notion of perfect filtering is associated with the DDM region overlaps. However, DDM regions are d-rectangles [7] and they usually do not accurately represent the actual filtering criteria required by federates (i.e., detection range as an obstructed radius around a vehicle). Therefore, while the DDM filtering can provide an efficient culling of most of the irrelevant data, federates must perform additional filtering of received data.

Supporting perfect filtering with Data Distribution Management requires pervasive changes to an RTI implementation. So the simple requirement that a fully compliant RTI support DDM generally adds additional overhead. Unlike the other services, there is often little that the RTI can do to avoid that overhead. Thus, if the federation does not use DDM and the RTI offers the ability to disable DDM completely, it is generally a

worthwhile configuration choice.

3 Impact of RTI Implementation Choices

Within the HLA framework and the bounds of the chosen environment, different RTI implementations offer a wide range of features and performance characteristics. For example, when designing an RTI to run over Ethernet, there is little choice but to use some combination of UDP and TCP to implement the underlying communication between federates. But the implementor is free to choose what bookkeeping information is communicated and when. Similarly, all RTIs must provide a system for generation of unique handles, but the implementor is free to choose an appropriate method. A shared memory implementation may simply use a combination of process id and timestamp, while a networked RTI may choose to provide a central handle generation service.

A federation designer must gage the limitations imposed by the specification and the chosen medium and then analyze the features of the various RTIs based on that understanding.

3.1 Network Overhead:

Simulations often have stringent upper bounds on the bandwidth they may use, so minimizing required bandwidth was one of the driving goals of HLA. First and foremost, bandwidth is a function of simulation design. The more interaction required between distributed simulation entities, the more bandwidth consumed. However, RTI implementations vary widely in the amount of bandwidth consumed per operation and even within an implementation some configurations may consume far more bandwidth than others.

When estimating bandwidth requirements, simulation designers often compare the structure of HLA's generic handle/value pair scheme with packed PDU formats such as DIS. But a direct comparison is misleading.

While predefined PDU may certainly be smaller than an update containing the state of all attribute instances of the equivalent HLA object, an HLA simulator is rarely required to provide all attributes in each update. Taking into account the fact that static objects and attribute instances require no updates under HLA while the entire state must be sent for each update in a fixed PDU format, the savings can be large. So while HLA requires a significant amount of overhead per attribute, the number of attributes is significantly decreased. In a protocol which must send the entire PDU each time, the unchanged data is overhead.

In terms of overhead bandwidth, the most important question to ask of an RTI implementor is “how many copies of each message will be placed on the network?”

Simple federations running entirely on the LAN may use multicast for all communication and place exactly one copy of each message on the wire. However, larger HLA federations rarely run entirely on the LAN and even within the LAN there are very good reasons to use Reliable transport for at least some messages. Due to the point-to-point nature of TCP, RTIs which use TCP for Reliable transport must send at least one copy of a reliable message per destination.

In many cases, an RTI may send more than that minimum number of copies. For example, an RTI may not filter at the sender based on subscription; instead it may simply send a copy of each message to each possible receiver. If an RTI employs a TCP exploder to reduce network computation costs at the sender at the cost of marginally increased bandwidth and latency, it will send an initial copy of the message to the exploder. When sending data across the WAN, even best-effort messages are likely to result in multiple copies per send since multi-cast is not well supported over a WAN.

Most RTIs offer several means for the federation to control the number of message copies. Federation designers should work with the implementors of the chosen RTI to optimize the number of messages on the network.

3.2 Asynchronous Operation:

RTI implementations also offer several methods of handling network I/O and issuing callbacks to the main simulation. The three basic strategies are single-threaded, evoked callbacks, evoked callbacks with asynchronous I/O, and fully asynchronous callbacks [8]. Offloading network I/O to a separate thread offers several key advantages. Most importantly, it can offer more immediate response to network events - leading to fewer dropped messages in "bursty" networks. In the case of fully asynchronous callbacks, the federate ambassador callback may also be called sooner relative to the evoked methods.

However, there are significant costs associated with asynchronous handling as well. In particular, asynchronous I/O requires a complete copy of all incoming and possibly outgoing messages, since they must be queued until the asynchronous handler is given processor time. This leads to additional latency for all messages. The fully asynchronous callback mechanism generally avoids making a copy on then sending side, but the receiving side is more complicated.

When comparing the performance of the three callback modes, users often attempt to measure as statistic know as "callback latency." Callback latency is the absolute time from a message being received on the network to the time that the RTI invokes the Federate Ambassador Callback method. Approaching the question in this way leads to results that are less useful than they appear. There are 2 really important questions here:
a) How much processor time is needed for a message to pass through the RTI to the callback?
b) What is the latency from the message reception to the simulation using the data?
Callback latency generally fails to measure either of these directly.

Federates which use the RTI in single-threaded evoke or evoke with asynchronous I/O modes demonstrate much larger callback latencies than federates using fully asynchronous callbacks, since callback latency is heavily affected by the rate at which a federate evokes callbacks in these modes. However, in most cases, the perceived latency savings are an illusion. The actual processor time spent reading a message and executing the resulting callback is actually minimized by the single-threaded callback mode.

Asynchronous callbacks do allow the RTI to make optimal use of the processor by offloading message decoding to a secondary thread that can use free CPU cycles. In some federates this can be a real advantage, particularly on multiprocessor machines. It is true that a callback may be executed earlier if it is allowed to happen asynchronously. But, since most federates expect simulation data to be stable for large portions of each frame, the asynchronous callbacks are generally designed to merely queue the data that they receive. If the federate allows a callback to update federate state asynchronously, then it must perform expensive locking to ensure data integrity. In either case, asynchronous callbacks require carefully crafted, multi-threaded federate applications to ensure that the concurrent execution does not cause unexpected or illegal behavior.

It is important to recognize the real costs and benefits of asynchronous and synchronous modes of operation and make the choice appropriately.

3.3 Scalability:

Perhaps the most important differentiator among RTI implementations and the most difficult performance factor to measure is scalability. The algorithms chosen by the RTI implementation have a tremendous impact on how performance scales as federation complexity grows. Different algorithms work well for different federation sizes and levels of interaction. Worse, it can be difficult to predict whether the results with a smaller federation are optimistic or pessimistic for the RTI under test. For example, there exist centralized time management algorithms for which, in the one and two federate case, it is possible for the LRCs of time managed federates to provide predictive time advance grants without waiting for confirmation from the central time manager. For RTIs applying these optimizations the performance of time management in the 2 federate case is unrealistic for larger federations. Other algorithms may be designed to accommodate large federations and may even have worst case performance in the 2 federate case. In RTIs using a central forwarder for message transmission, there is always one additional hop / message copy for each message. Thus, in the two federate case, a forwarding architecture will require fully twice the bandwidth and latency of a fully connected network. But, as the federation grows that percentage overhead decreases proportionally.

A few of the key measures of scalability that apply across RTIs and federations are: centralized processing load, message volume, and local RTI processing.

First, it is important to assess the proportion of RTI processing that occurs at a single central point. For very large federations, centralized processing is a limiting factor since the processor running the central component is fixed as federation size increases. A centralized algorithm may actually be optimal for some services and for federations up to a certain size. However, to make these algorithms scalable, the RTI must offer methods of distributing the load of central processing as the federation grows.

Second, it is useful to examine the volume of messages generated by the RTI as federate count increases. RTI implementations generally attempt to offer messaging scalability that is much better than linear in number of federates. For example, most RTIs implement best-effort using UDP multicast, so for best-effort updates and interactions only one copy of the message must be transmitted independent of federate count. As mentioned before, reliable transport generally scales linearly in number of federates – requiring at least one message copy per subscribed federate. Most RTIs offer solutions to this problem (for example, an RTI may allow federations to reduce the local message count by separating federates into subnetworks) and federations should be designed to take advantage of these solutions.

Finally, it is critical to examine the LRC's CPU utilization as federate count increases. As with message count, if the LRC requires increasing CPU time to send messages and

process callbacks as federate count increases this may limit scalability. Additional message load will increase CPU and bandwidth utilization in any RTI implementation. However, in some RTIs there may be costs associated with simply adding a federate. Consider a two federate case with a single publisher generating updates at 10 Hz and a single subscriber, if a second subscriber is added to the network the number of updates sent by the publisher is constant. If the publisher's LRC requires additional processing time as subscribers are added, then there may be a scalability issue that will affect the target federation as well.

3.4 Data Distribution Management algorithms

The Data Distribution Management services are perhaps the most complex features of HLA. DDM has the potential to greatly increase the scalability of a federation. But to be effective, the use of DDM requires intimate knowledge of the DDM implementation of the chosen RTI. Federations optimized to take advantage of one DDM implementation may find that DDM adds large amounts of overhead with little gain in terms of filtering when they switch RTIs.

For example, two DDM algorithms, distributed regions and fix-grid, in current use in different RTIs make very different trade-offs. A distributed region scheme inherently distributes the region information and performs region overlaps using the federate update and subscribe regions, or an amalgamation of them, to establish data routing [2], [6]. A fixed-grid scheme performs region overlaps using an a priori fixed grid that establishes data routing [3]. Given these two DDM representations, there is a significant difference in how the federation will perform depending on how the DDM is incorporated into the federation design.

The distributed region approach will inherently support perfect filtering and advisories. However, if used inappropriately, the exchange of region information can easily overwhelm any savings provided by the DDM filtering. This approach is optimal for federations that use regions with infrequent changes [5].

The fixed-grid approach need not exchange any region information in order to perform data routing. The fixed-grid establishes data routing through assignment of multicast groups to the grid cells. To route data, the federate regions are then overlapped with the fixed-grid. In this scheme, region modifications are a strictly local operation and can be performed frequently. This provides very good scalability. However, the fixed-grid approach does not perform perfect filtering nor does it support advisories without a great deal of additional overhead. If perfect filtering and advisories are required, then the federation would be better served by the distributed region approach.

4 Impact of Federation Requirements

Part of the reason that configuring an RTI is difficult is that the factors which affect the performance of a given implementation are not obvious. Federation developers often ask questions like “How many objects can this RTI implementation handle?” “How many federates can it accommodate?” “How does performance scale as the federation grows?” Implementors invariably answer “That depends.” Here are some of the reasons that those aren't easy questions to answer.

4.1 Number of objects

HLA objects can be very lightweight so for most implementations, federations must contain many thousands of objects before the number of objects becomes an issue. When numbers of objects grow into the hundreds of thousands, the sheer number of objects might start to be the limiting factor, due to memory usage, lookup efficiency, etc. However, most federations will hit other performance limitations long before number of objects per se becomes a factor.

Regardless of the number of objects, the rate of updates is one of the primary performance limits. It matters very little if a federation contains 10 objects updates 100 times per second, or 1000 objects updated once per second. Certainly, the specification imposes very little direct overhead on attribute updates based on the number of registered objects.

The RTI must maintain some state for each object, and it is important to recognize that this state is purely overhead from the federate's point of view. The object state maintained by the RTI is on top of the state required to simulate the objects. This overhead scales linearly with the number of objects. Different implementations will use differing amounts of memory, but a sample implementation of an HLA object instance at an LRC may consist of no more than the following:

Object Instance Field	***Type***
Object Instance Handle	ObjectInstanceHandle
Object Instance Name	String
Object Class Handle	ObjectClassHandle
Object Class Name	String
Discovered Class Handle	ObjectClassHandle
Discovered Class Name	String
Owned Attributes	AttributeHandleSet

Object Instance Field	*Type*
Attributes In Scope	AttributeHandleSet
Relevant Attributes	AttributeHandleSet

Plus some additional state per attribute instance of the object:

Attribute Instance Field	*Type*
Transportation	TransportType
Order	OrderType
Associated Region Set	RegionHandleSet

An RTI may also associate other fields with the object, such as a set of unowned attributes, the handle of the owning federate, the set of attributes pending ownership transfer, and possibly a list of subscribers to allow sender-side filtering.

The important point is that the RTI has no requirement to store the actual attribute values, which tend to be large. So while the simulation will need to maintain at least one copy of the complete object state, which is likely to consume a large amount of memory, the memory footprint per HLA object is very small by comparison.

Of course, in any federation, the number of simulation objects matters. But, the number of simulation objects matters primarily because of the cost of simulation of the objects and simulation of interactions between them. The overhead added by the RTI per object is rarely a primary cause of performance problems.

4.2 Number of federates

Federation developers are also often concerned about the maximum number of federates connected to a federation. Scalability is a very important measure of an RTI. But as with HLA object instances, in a well implemented RTI, the number of federates matters far less than the amount of work performed (in particular messages sent and received) per federate. For example, a federation consisting of hundreds of federates simulating primarily static objects and just a few moving objects is likely to have fewer performance problems than a federation of 3 federates each simulating thousands of infantry interacting with each other.

Unlike the object case however, number of federates matters greatly in certain cases. Reliable transport is the most obvious case. Since Reliable requires an individual copy of the message to be sent to each federate, in a federation with 1000 federates, to update 1 object, 1000 copies of the update must be sent. In a federation of 10 federates, just 10 copies are required. If those updates were sent best-effort only one copy would be sent in either case. Thus, the ratio of Best-Effort to Reliable data is a key factor in determining scalability. Similarly, if the federation is time managed, the synchronization problem becomes harder as the number of federates increases.

In most cases, finding the optimal break down of simulation tasks between federates is generally more important than limiting the number of federates. The answer to this question depends on the ratio of best-effort to reliable traffic required, whether the federates must be synchronized using time management, and most importantly the simulation tasks of each simulation entity. It also depends on the options provided to the federation by their chosen RTI.

4.3 Disabling Services:

In the past, it has been common in federation agreements to disable certain HLA features altogether. Most current implementations offer techniques to disable service categories such as Time Management, DDM, or the MOM and features such as Reliable transport. Several existing RTIs offer a fully distributed mode of operation, sometimes known as “connectionless mode” which automatically disables several core services and limits the federation to best-effort transport. Some of the reasons which lead to those federations decisions no longer apply and it is important to re-examine the benefit achieved in return for this loss of functionality.

Reliable transport, in particular, offers important advantages (even within the LAN) for large federations. Most importantly, reliable messaging allows the RTI to make guarantees about the state of the federation. Some RTIs have gone to extreme lengths to ensure that the federation can operate stably even if critical internal messages are lost. For example, RTIs may allow “lazy discovery” of objects – if the initial object registration is lost, federates can discover the object based on the first update they receive. RTIs may even offer the ability to heartbeat objects to ensure that static attributes are eventually reflected at remote federates. These compromises eliminate many of the benefits of HLA discussed previously. Other services such as time management are nearly impossible without reliable transport. Reliable transport allows the RTI to eliminate heartbeats for discovery and allows federations to lower bandwidth consumption by sending static attributes just once.

As mentioned in the discussion of time management, disabling core RTI services generally offers less benefit than newcomers to HLA might expect. HLA is designed so that for the most part federates only pay for the

services they use.

In general, RTI configuration is more effective when it simply relaxes a “shall” in the specification rather than eliminating a service altogether. For example, RTIs may offer a mode that disables perfect DDM filtering which eliminates the need to perform "perfect" region based filtering. Such an option can lower the cost of DDM significantly while still providing a useful form of data scoping. Similarly, if the specification of Reliable transport is relaxed to allow federates experiencing faults to drop messages, then the RTI may be able to provide a much less expensive form of fault recovery. There are many other options available.

4.4 Federation Specific Constraints:

Finally, it is common for a federation to have hard constraints. Sometimes these constraints will be the prime factors in the choice of RTI or HLA in general. For example, if the federation must run on a specific clustering hardware, there may be only a limited number of RTIs available which offer support for that platform. Similarly if the simulation has hard-real-time requirements then only an RTI designed to support real-time guarantees will be acceptable, and since the HLA API is not directly targeted at hard-real-time operation it is possible that the federation will only be successful with a different distributed architecture.

More commonly, there will be multiple RTIs which support the federation but the federation constraints may still make the choice of RTI simple. As mentioned earlier, number of objects is rarely a limiting factor. But if a federation is simulating hundreds of thousands or millions of objects, then it may find that current hardware limits mean that even a few bytes difference per object make a difference. Similarly, if an RTI must operate in a heavily bandwidth limited environment (a ship-to-ship link for example), then ability to control and limit bandwidth may be the dominant factor. If an RTI implicitly violates a hard constraint then it can easily be eliminated from consideration.

One operational constraint that almost all current federations face is network security. Often a federation with connections across the WAN will be required to operate according to security rules beyond the designer's control. When operating across firewalls, RTIs may pose various difficulties. For example, when crossing a secure boundary, number of connections, addresses and ports are all likely to be limited. An RTI which offers only a fully connected network approach will be unlikely to meet those goals. The HLA API is flexible enough to allow these issues to be handled beneath the API with no special consideration from the federate. Most RTIs offer a means of firewall traversal, such as a forwarder to provide a single point of ingress and egress across the firewall. RTIs may offer additional services such as filtering of secure data, or encryption techniques. These options may reduce the need for specific security hardware.

5 Impact of Physical Resource Constraints

The performance constraints imposed by the physical resources available apply to all federations regardless of RTI. Latency, bandwidth, processing power and memory are the most basic of these. Large federations commonly push the limits of their environment and in doing so they must make trade-offs between limiting factors. Each RTI offers different methods of dealing with specific physical constraints.

It is critical to compare RTIs and RTI configurations in the context of the physical resources available. Only in that context is it possible to measure the success of a new configuration. Imagine an operating system in which a simple TCP socket send call takes 20 µs of CPU time. If you find an RTI that executes an updateAttributeValues call in less than 20 µs, you can deduce that it must be delaying the actual network call until after the function returns (for example, by using bundling or asynchronous I/O). Of course, delaying the actual network send introduces additional latency. By supporting both an immediate-send version and a delayed-send version of updateAttributeValues, an RTI offers the federate developer a trade-off between CPU usage and latency.

Configuring any RTI implementation for optimal performance requires making such trade-offs between CPU and memory usage, latency, and bandwidth. Because the optimal trade-offs vary according to federation needs, RTI implementors cannot provide an out-of-the-box configuration which is best for all users. Instead, the implementors either optimize for a specific use or attempt to provide an out-of-the box configuration which offers reasonable performance and usability to their intended users.

The following sections attempt to describe some of the resource constraints and the sorts of flexibility RTIs are able to offer within them.

5.1 Latency and Per-Update Processing:

For most federations, the most important indicator of RTI performance is the maximum update rate achievable for the federation's typical message sizes and subscriber counts. Both end-to-end latency and the amount of CPU time devoted to each update and reflect play significant roles in federation performance. Latency and update rate

are both relatively simple characteristics to measure if the federation designer can estimate the appropriate message sizes and federate counts. The callback rate is a much less easily quantified. Callback processing is largely a function of the federate specific callbacks and can be controlled further via the evokeMultipleCallbacks timeouts. In most cases, the RTI processing per update can be used to estimate the internal RTI processing per reflect since they are often symmetric.

End-to-end latency is certainly the most common benchmark performed by HLA users, and update and reflect processing are components of end-to-end latency [9], [10], [11], [12]. However, the latency measurement alone often fails to encapsulate the RTI overhead at the sender and per update processing by the RTI is often the greatest impact of RTI implementation on federation performance. All local processing performed by the RTI is the time taken away from simulation work in each frame.

Per-message processing is such an important contributor to overall performance that many RTI implementations offer means to trade-off increased end-to-end latency for reduced processing at the LRC. The use of a TCP exploder for reliable data rather than performing all sends locally is one example. RTIs may make this trade-off by centralizing other algorithms as well. For example, if an RTI offers a centralized approach to DDM filtering, it may require each update to make an additional network hop to a DDM proxy. This will probably more than double the latency for each update and compare very unfavorably in direct latency tests. However, if the removal of DDM processing from the LRC provides a significant reduction in latency from the update call to the network send, then in an active simulation such an algorithm may provide a substantial performance gain. Of course, unless handled carefully, any algorithm that centralizes work may lead to the scalability issues discussed later.

Portions of the per-update processing can be controlled by the federate and federation. In a common example, the federate can reduce the work spent generating the handle/value pairs by caching the handles so that lookups are not required per frame. The HLA APIs are also designed to allow the user and the RTI to reduce the number of unnecessary copies of data in memory. For example, in the C++ API, the VariableLengthData type has two accessors to the contained data: setData and setDataPointer. Use of the first (or the corresponding constructor call) is often more convenient to the caller, but use of the setDataPointer method allows the RTI to avoid making an internal copy of the variable length data. In C++ federates, the number of copies of attribute and interaction data which the RTI must create is a key performance concern.

It is also possible to reduce per update/interaction processing through optimization of the FOM. It is certainly more efficient in terms of overhead, and likely the amount of work done internal to the update call, to pack all attributes of and object class into a single structure represented as a single FOM attribute. However, this can have an unexpected negative impact on overall federation performance. Doing so eliminates the opportunity to reduce bandwidth consumption by sending only the changed attributes (unless the federates replicate the functionality already provided by the RTI). Worse, doing so negates the RTI's ability to perform sender-side and receiver-side filtering based on partial subscription interests. Thus, attempts to optimize HLA usage through redesign of the FOM should only be done after careful analysis of the data requirements of all federates. A example of how this can be effective is the SpatialStruct in drafts of the RPR FOM version 2.0 and later [13]. Since federates interested in any portion of the Spatial data are likely to be interested in all elements, there is unlikely to be any gain from subscription filters.

5.2 Bandwidth and Throughput:

Bandwidth is perhaps the most straightforward measure of performance in any distributed system. The two basic measures are the average load on the distribution system and the maximum achievable data rate. Measuring bandwidth utilization is straightforward, but the measurement should be made in the context of the real simulation system. In particular, any bandwidth benchmark needs to emulate the ratio of best-effort and reliable traffic of a given federation, and the ratio of publishers to subscribers. The benchmark should also attempt to represent the targeted network topology. Specific RTIs may have offer other criteria which can affect bandwidth usage as well.

Emulation of the target federation's use of the RTI is critical because different RTIs may offer very different bandwidth performance in different environments. As mentioned in the discussion of reliable and best-effort transport, reliable transport generally requires multiple copies of the message to be sent (usually one copy per destination or more) while best-effort may require only copy to be sent if multi cast is used. So it is clear that the higher the ratio of reliable traffic to best-effort, the more bandwidth used. Less obvious is that most RTIs offer specific optimizations for operation across a WAN and these may have a large impact on bandwidth usage in federations which will be distributed across multiple LANs. These optimizations may be aimed at reducing the impact of the WAN on LAN performance, or reducing the amount of data sent from the high bandwidth LAN onto

the low bandwidth WAN link. For example, an RTI may offer a built-in tunneling solution to ensure that only one copy of any message passes between any two LANs. Such a solution is designed to prevent the LAN from flooding the WAN link, so when it is applied the bandwidth on the WAN should be lower. But if there is an additional network hop introduced by the tunneling application total bandwidth usage will still be higher than for a federation entirely on the LAN.

The other key issues to identify in the benchmarking process are any RTI algorithms that may have unexpected costs in terms of message passing. Using HLA in any distributed system requires the exchange of internal bookkeeping messages each of which takes a measurable time and bandwidth, and therefore limits the number of operations a federate may perform. If the federation will use time management or DDM in particular, it is useful to examine the number of messages passed by the RTI's time management and DDM algorithms. Often an RTI will choose an algorithm which requires more messages to be exchanged in return for some other benefit such as more efficient time advancement or perfect DDM filtering. It is then up to the designer to decide whether the trade-offs chosen by the RTI fit well with the goals of the federation.

All current RTI implementations offer bandwidth reduction techniques. The more configurable the RTI by the user, the more the designer is able to choose the proper trade-offs. RTIs generally offer message based solutions such as message bundling and compression which trade-off latency and CPU utilization for lower network utilization. An RTI may also offer some form of traffic shaping such as delayed transmission of subscription/publication/registration messages or rate limiting for updates and interactions. The first option minimizes the number of messages required at startup time and to support late joiners at the cost of additional startup delay for some federates. The second reduces bandwidth almost uniformly across the federation at the cost of additional message loss. Another common area of optimization is update relevance filtering beyond the basic publish and subscribe mechanisms of HLA. For example, in cases where the network is overloaded, the RTI may allow unsent updates to be aged and replace older updates of the same attribute instances with new values – thus avoiding at least one update altogether at the cost of losing the intermediate state.

These are only a small sample of the options available, and the various RTI implementations are adding new optimizations all the time.

6 Summary

One of the primary advantages of an API standard like HLA is that each implementation is free to offer different features and trade-offs, and competition ensures that the implementations are evolving all the time. Federations can take advantage of these features to improve performance throughout their lifetime simply by modifying the RTI configuration or possibly even switching RTI implementations based on changing requirements (generally with no additional federate development required).

Reaching the performance goals of a federation often requires designers to have a deep understanding of HLA and the issues driving RTI implementation. By controlling the behavior of the RTI and taking advantage of vendor specific features the designer can force the distributed system to make trade-offs appropriate to the current federation. The HLA specification, the chosen implementation, the available resources, and the federation architecture will all play roles in determining the correct trade-offs. What really matters is that federation designers understand how these elements interact and make the trade-offs based on that understanding rather than trial and error.

7 References

[1] IEEE: "High Level Architecture (HLA) – Federate Interface Specification", September 2001

[2] D. Wood: "Implementation of DDM in the MAK High Performance RTI" Simulation Interoperability Workshop, paper 056, Spring 2002

[3] M. Hyett, R. Wuerfel: "Implementation of the Data Distribution Management Services in the RTI-NG" Simulation Interoperability Workshop, paper 091, Spring 2002

[4] K. Morse, F. Hodum, P. Wickes: “Attribute Level Advisories and Scalability” Simulation Interoperability Workshop, paper 066, Fall 2001

[5] K. Morse, L. Bic, M. Dillencourt: “Characterizing Scenarios for DDM Performance and Benchmarking RTIs” Simulation Interoperability Workshop, paper 54, Spring 1999

[6] D. Van Hook, J. Calvin: “Data Distribution Management in RTI 1.3”, Simulation Interoperability Workshop, paper 206, Spring 1998

[7] M. Petty: “Experimental Comparison of d-Rectangle Intersection Algorithms Applied to HLA Data Distribution”, Simulation Interoperability Workshop, paper 016, Fall 1997

[8] M. Karlsson, P. Karlsson: "An In-Depth Look at RTI Process Models", Simulation Interoperability Workshop, paper 055, Spring 2003

[9] D. Nemeth: "Benchmarking the RTI for Simulated

Radio Applications", Simulation Interoperability Workshop, paper 046, Spring 1999
[10] T. Burks, T. Alexander, K. Lessman, K. LeSueur: "Latency Performance of Various HLA RTI Implementations", Simulation Interoperability Workshop, paper 015, Spring 2001
[11] P. Knight, R. Liedel, M. Klinner, J. Steele, R. Drake, P. Agarwal, E. Núñez, M. Espinosa, J. Giddens, C. Jenkins:, "Analysis of Independent Throughput and Latency Benchmarks for Multiple RTI Implementations", Simulation Interoperability Workshop, paper 068, Fall 2002
[12] R. Drake, P. Agarwal, M. Espinosa, C. Owens, R. Liedel, J. Steele: "Independent Benchmarking of RTI Real-Time Performance", Simulation Interoperability Workshop, paper 024, Fall 2003
[13] B. Murray, A. Faier: "Designing FOMs for Performance", Simulation Interoperability Workshop, paper 116,

Author Biographies

BEN WATROUS has been a key member of the MAK RTI development team for over 3 years. He has lead the development of RTIspy; creating extensions to the RTI's plug-in capabilities and graphical user interface. He has also been responsible for optimizing the RTI for use over Wide Area Networks, including design and implementation of the Distributed TCP Forwarder approach. Mr. Watrous has over eight years of experience developing distributed computing applications. Outside of MAK, Mr. Watrous worked as the lead designer for a distributed architecture for rendering 3D graphic images. Mr Watrous holds a Bachelor of Science and Master of Engineering in Electrical Engineering from Cornell University.

DOUGLAS WOOD is a Principal Software Engineer at MÄK Technologies. Mr. Wood is leading the development of the MAK High Performance RTI. Over the past 18 years, he has performed research and software development in distributed simulation, including DIS-HLA translation, electronic warfare protocols, emergency management training and computer generated forces. Mr. Wood received an MS and BS in Computer Science from the University of Central Florida. His interests are in simulation and distributed systems.

LEN GRANOWETTER is the Director of Product Development at MÄK Technologies, responsible for MÄK's COTS product development group. He was the chief architect of the MÄK High-Performance RTI during its initial development. He has been involved with MÄK's distributed simulation products for over 13 years, including as MÄK's Lead Products Engineer for several years during the HLA transition. Mr. Granowetter holds a Bachelor of Science degree in Computer Science and Engineering from the Massachusetts Institute of Technology.