RTI Interoperability Issues – API Standards, Wire Standards, and RTI Bridges

Len Granowetter
MÄK Technologies
185 Alewife Brook Parkway
Cambridge, MA 02144
617 876-8085 x121
lengrano@mak.com
http://www.mak.com

Keywords:
HLA, RTI, Interoperability

ABSTRACT: *With the proliferation of commercial Run-time Infrastructures (RTIs) for the High-Level Architecture (HLA), issues regarding RTI interoperability have taken on new importance. This paper delves into some of these issues, and discusses various approaches to succeeding in a world with multiple RTI implementations. At times, members of the HLA community have suggested the development of several levels of RTI Application Programmer's Interface (API) standards, network-level "wire standards" for RTIs, and RTI bridging technologies as possible techniques for dealing with multiple RTIs. We will use our perspective as a developer of federates that need to support multiple RTI implementations, as an RTI developer, and as an RTI-bridge developer to describe our experiences and opinions regarding these techniques.*

1. Introduction

In September of 2002, we entered a world where multiple RTI implementations are a fact of life for HLA developers. While the presence of multiple RTI implementations gives HLA developers new choices, it also creates a new barrier to interoperability: Federates typically cannot interoperate unless they are all using the same RTI implementation.

The question facing HLA developers today is how to deal with this new interoperability barrier.

Some have advocated trying to *eliminate* the barrier altogether by developing "wire standards" for HLA. If different RTI implementations could all speak the same language, federates would be able to communicate even if they were not using a common RTI implementation.

A second approach is to *work around* the interoperability barrier, by creating an RTI-to-RTI bridge. Such a bridge would "translate" between two RTI implementations, allowing federates connected to different RTIs to communicate.

A third option, is to in effect *lower* the barrier imposed by the restriction that all federates in a federation must use the same RTI implementation, by making it easier to switch among different RTI implementations. After all, if switching to another RTI implementation were made simple enough, the fact that all federates must agree on a single RTI implementation to play in a given federation execution would not necessarily seem like a very significant barrier at all. (Obviously, lowering the barrier is more than just a technical problem).

We intend to show in this paper, that the third option – making it easy to switch among RTI implementations - is the one where we should be focusing our efforts to overcome the interoperability barrier. We will show that a wire standard for HLA, while a tempting solution, does not make sense once one actually tries to define what it might look like. Similarly, we will describe the limitations of the bridging concept as applied to HLA, and why an RTI-to-RTI bridge that covers all RTI services is simply not possible. Finally, we will show how progress in the area of RTI dynamic-link-compatibility standards is already making it trivial for many federates to switch among several RTI implementations, and how future effort can lower the interoperability barrier even further.

2. The Allure of Wire Standards for HLA

Many in the HLA community have previously worked with communication methodologies such as Distributed Interactive Simulation (DIS), where standardization is enforced at point where data is written to, or read from, the network. The DIS standard dictates exactly how various elements of simulation data are to be encoded within UDP packets, so that remote applications can

understand them when they are received. Packet layout, byte ordering, data representation, and rules for deciding when various kinds of packets should be sent, are all dictated by this "wire" standard (so named because it regulates how data is placed "on the wire".)

HLA on the other hand, is a completely different kind of standard. Rather than dictating how data must be placed on the wire, HLA defines a standard API to a Run-Time Infrastructure that a federate uses to communicate with other federates. On the sending side, a federate passes simulation data to the RTI through the standard API, and on the receiving side, the RTI delivers that data to other federates through the standard API. What happens in between depends on the particular RTI implementation one is using.

A wire-standard-based protocol has several significant advantages over HLA, and one of the most important is that it does not require applications to share any common software modules. A federate developer can choose to write every line of code by himself, and still be able to interoperate with others. He is not constrained by the platform or language choices of an RTI developer, or anyone else. As long as he can get his data on and off the network, he can play. By contrast, in HLA, all federate developers in a federation must agree on a single RTI implementation. This may be problematic, especially in cases where no single RTI implementation supports all of the federates' desired platforms or language bindings.

It is only natural that users of HLA would want to regain some of the freedom afforded by wire-protocol-based standards, and on the surface, it seems like this could be achieved by defining a wire standard for exchanging RTI-internal data. Federates would still write to the API defined by the HLA Interface Specification, but now a federate developer also write his own implementation of his Local RTI Component – mapping RTI calls to the standardized network packets.

Unfortunately, implementing this approach would eliminate many of the key *advantages* of HLA. We are not arguing that HLA is a better or worse approach to simulator interoperability than a wire-protocol-based approach; a full comparison of the advantages and disadvantages of each is well beyond the scope of this paper. What we *are* arguing, is that a wire standard underneath HLA does not make it possible to get "the best of both worlds."

Unfortunately, one of the most common arguments we have heard for why a wire standard for HLA does not make sense is that different RTI implementations may not even use the same transport mechanism. One RTI might use a TCP/IP, or IP/UDP-based network, and another might use shared memory, for example. However, proponents of an HLA wire standard are rightly unconvinced by this argument. "Why can't we at least have a wire-standard for the most common variety of RTIs – those that use standard TCP/IP or IP/UDP networking?" they ask.

The answer to that question is that even within a given type of communications framework, different RTI implementations differ in much more than just the way they choose to lay out their packets. Decisions about what kinds of packets an RTI needs to exchange, what data needs to be encoded in those packets, and even where and when packets need to be sent are intimately tied to top-level design choices made by each RTI's developers.

Specifying a wire protocol for HLA is tantamount to mandating a specific DDM algorithm, a specific Time Management algorithm, a specific approach to approximating reliable multicast on top of a unicast protocol like TCP, and a specific strategy for distributing RTI advisory information. The problem with this is that there is no single approach to any of these issues that is clearly superior to all others. Different RTI implementations have tackled each of these problems differently, and each approach could be considered superior, depending on the requirements of a federation.

Diversity in RTI implementations is a key advantage of HLA. Rather than have to settle for a lowest-common-denominator approach to interoperability, each federation can choose an RTI whose design tradeoffs best match its requirements. One RTI's DDM implementation might work best for federations where regions do not change very often, while another might work best for federations where regions are constantly being updated. One RTI might be most suitable for federations where throughput is more important than latency, and another might be best for a federation where low latencies are the top concern.

The problem is not that a wire standard for HLA would be constraining to RTI developers, as some argue. It's that it would be constraining to federation developers who would no longer have the freedom to evaluate tradeoffs between different technical approaches.

One example of a design tradeoff faced by RTI developers is how to send data from one federate to many federates using a reliable transport mechanism. TCP is the most common underlying reliable transport protocol, but by itself, TCP allows an application to send a packet to only one recipient at a time. And unlike UDP, TCP requires the establishment of a connection before any data is exchanged.

Two common approaches to achieving the ability to send to multiple federates using TCP are a fully-connected network, and a TCP exploder. In a fully-connected network, a separate TCP connection is established between each pair of federates. A sending federate can send a copy of each packet directly to each remote federate by making a separate send call on each TCP socket. In the TCP exploder approach, each federate maintains only a single TCP connection to a central server known as a TCP exploder or TCP forwarder. A sending federate sends a single copy of the packet to the exploder, who forwards a copy to each remote federate.

The fully-connected network approach often minimizes latency (of reliable packets), because a packet takes only one hop between the sender and receiver, as opposed to two hops for the TCP exploder approach. However, the TCP exploder approach usually minimizes the amount of time the federate spends in a call to updateAttributeValues() or sendInteraction(), because those calls can return after sending a single copy of each packet, instead of having to wait until a copy is sent to each federate. So which is more important? It depends on your federation's requirements.

But let us assume for the sake of argument that a wire standard for HLA dictated the use of the fully-connected network approach. The next design decision is how the connections are to be established. Does each federate initiate the connection to all federates that have already joined? Does each existing federate initiate the connection to the late joining federate? How does each federate find out the IP addresses of the federates it needs to connect to? Does the RID file dictate this? Does it listen for an announcement by the joining federate? Is it told by the rtiexec? Again, there are tradeoffs between ease of configuration, bandwidth required, and the amount of time it takes to initialize a federation. Which approach is better depends on your perspective.

A second example is the implementation of Data Distribution Management. Here, two common approaches are the distributed region approach, and the fixed grid approach [1]. In the fixed grid approach, associations are made between grid cells and multicast addresses in advance of federation execution. In the distributed region approach, these associations must be communicated among Local RTI Components (LRCs) during run-time. However, the distributed region approach allows for more accurate multicast filtering, and does not ever require a packet to be sent multiple times to different multicast addresses, as can happen with fixed-grid DDM. Depending on how many regions a federation has, how often they are changing, whether publication regions are point-regions or not, and how many updates and interactions are likely to be sent to each region, either one of the two competing approaches could be considered superior.

So, even if we make the assumption that all RTIs are using UDP multicast for DDM, should our wire protocol dictate that we send region/address associations at run-time? No matter which choice we make, we will be locking all RTI implementations into a single design choice for DDM, and prohibiting federation developers from choosing the design that best fits their federation's profile.

Developing a wire standard for HLA is much more than just deciding how to lay out attribute handles and values in a packet. As a recent post to the SIW-RTI reflector put it:

I think the long-term cost of imposing a workable on-the-wire standard is way too high. There is still too much to be gained from having RTI developers look for better (and, almost by definition, probably incompatible) ways to make their RTIs more efficient, reliable, scalable, fault tolerant, and so forth. [2]

3. The Promise of RTI Bridges

In the simulation world, bridges and gateways are often the first things that come to mind when confronted with a question of how to link dissimilar systems. DIS-to-HLA Gateways are popular, and FOM-to-FOM bridges have shown some promise, so RTI-to-RTI bridges seem like a natural solution to the problem of linking federates that have been built against different RTI implementations.

The general approach is to build an application that can receive data from federates using one RTI implementation, and forward that data to federates using another RTI implementation. But because there is no wire standard for messages exchanged among LRCs, an RTI-to-RTI bridge cannot just listen to the network, and forward the packets that it sees. The only way for the bridge to receive data from, and send data to, the various federates, is through the RTIs' APIs. This means that an RTI-to-RTI bridge must actually join all of the federation executions that it is trying to bridge, acting as a federate in each (See Figure 1).

In principle, this blueprint for an RTI-to-RTI bridge sounds feasible, and in fact, for a certain subset of RTI services, it is. In fact, several working implementations of RTI-to-RTI bridges have been built and demonstrated, including one by MÄK Technologies, and another by the Software Engineering Institute at Carnegie Mellon University and SAIC [4], [5].

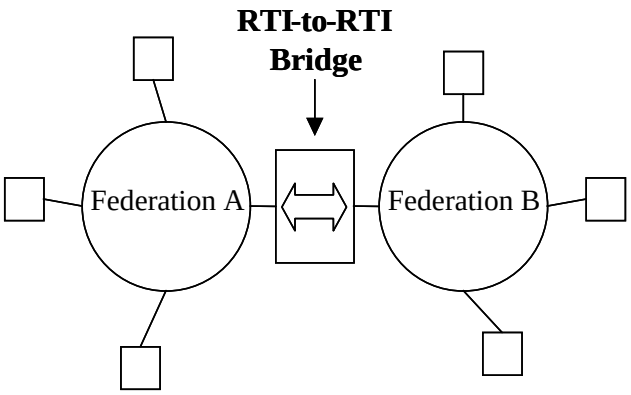

Figure 1: The RTI-to-RTI Bridge joins both federation executions, serving as a federate in each.

However, in addition to the obvious impact bridging has on performance, a lot of functionality is lost when using RTI-to-RTI bridging, and for some of the RTI services, RTI-to-RTI bridging is simply not possible. We will go through some specific examples that demonstrate why this is true, but the basic reason for this is that there is a lot of internal RTI state and functionality that is not available to the bridge through the RTI's API [3]. In other words, there are many services that a federate can invoke that change the state of the RTI or cause the RTI to do something, but that do not result in an RTI-initiated service being invoked on a remote federate such as the bridge. If a bridge has no way of knowing that a federate has invoked a certain service, it cannot invoke that service on the federate's behalf in the other federation executions.

For example, in DDM, when a federate subscribes to a certain interaction class with an associated region, there is no way for an RTI-to-RTI bridge to know this. When a federate associates a published attribute with a region, there is no way for the bridge to know that either. This alone renders DDM useless in a federation that is using an RTI-to-RTI bridge. Even if subscription and publication information were available, however, bridging between two different DDM strategies does not make sense. DDM is an optimization technique, the benefits of which can only be realized when the underlying communication mechanisms can be inspected and controlled.

Even for simple Declaration Management-based subscription, bridges have limitations. When a "listening" federate on one side of the bridge subscribes to a particular object class with a set of attributes, the bridge will want to cause a similar subscription to occur on the other side; otherwise, federates that can publish relevant data may never start generating updates. However, the bridge cannot directly detect that the subscription even occurred. A bridge can only *indirectly* determine that a federate has subscribed to the class by publishing the class, and waiting for an RTI advisory such as startRegistrationForObjectClass.

But even at that point, the bridge does not know what attributes the subscriber is interested in. The bridge needs to subscribe to some arbitrary subset of the object class's attributes to force the publisher to register the object. Upon discovering the object, the bridge can register a corresponding object on the receiver's federation, and wait for a turnUpdatesOnForObjectInstance() callback. Only then does it know what attributes the listener originally subscribed to, and can it finally make the (second) more accurate subscribe call on the listener's behalf. The order and timing of the various RTI calls certainly will not match what would have occurred had this been a more typical, single-RTI-implementation federation.

Ownership transfer across an RTI-to-RTI bridge is problematic as well. Although this functionality has been successfully prototyped [5], the authors point out that there are cases where multiple federates on different sides of a multi-federation bridge can both think they've acquired ownership of the same logical object.

Debugging of a federation is made much more difficult with an RTI-to-RTI bridge. Each logical object has to exist as a separate HLA object in each federation, and will have a different object handle in each. Logging and correlating corresponding events on each side of the bridge would be a daunting task.

Finally, a successful RTI-to-RTI bridge must re-create so much of the internal state of the RTI (often by gathering data through rather indirect methods) that its complexity starts to rival an RTI implementation itself. It would seem that the cost of building and maintaining a fully-functional RTI-to-RTI bridge, plus the cost of integrating with several RTI implementations and a bridge would far outweigh the cost of requiring all federates to agree on an RTI implementation.

We believe that RTI-to-RTI bridges are useful for situations like hierarchical federations, where a distinct separation between the federations on each side of the bridge is desired. (Perhaps the different federations are using different FOMs, or different levels or security). But if the goal is to emulate a situation where all federates are participating in a single federation, then an RTI-to-RTI bridge is a questionable approach.

4. The Reality of Non-Interoperable RTIs

As we consider the option of just living with the fact that different RTI implementation are not interoperable, it is important to review what an RTI really is. The term Run-

Time Infrastructure really does refer to the whole infrastructure needed to connect federates and allow them to share data. It is an implementation of the HLA Interface Specification that includes software libraries that are linked into each federate, plus any necessary supporting tools or executables. The rtiexec or other central server is not an RTI, and a Local RTI Component (LRC) – the piece of the RTI that is linked into each federate – is not an RTI. The RTI is the whole package (See Figure 2.)

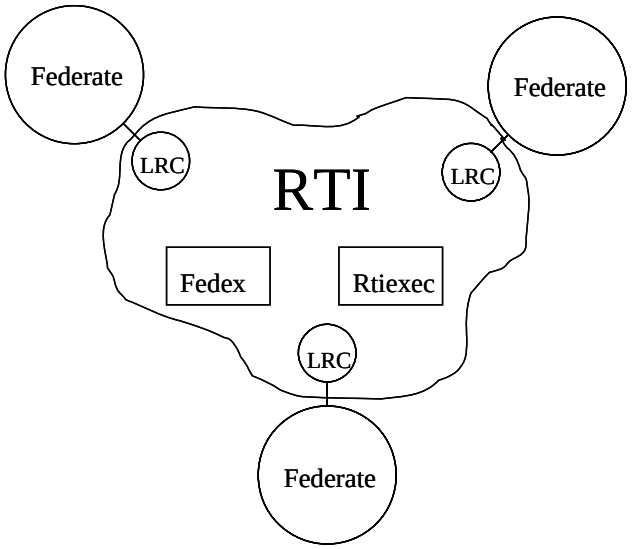

Figure 2: The RTI includes the Local RTI Components (LRCs) linked into each federate, and any other supporting software required by the implementation.

This is not an irrelevant semantic distinction. Recognizing this oft-forgotten basic premise of HLA makes it obvious that asking how federates can communicate while using different RTI implementations makes little technical sense.

The very idea of HLA was to define an interface that federates could use to exchange data, and to leave the implementation of that interface to the RTI. Because the interface is a standard, switching from one RTI implementation to another as one moves a federate from one federation to another *should* be easy. The choice of which RTI to use is a federation-wide decision, not a federate decision.

By way of analogy – we turn to another standard that exists at the API level, rather than at the level of data format: OpenGL. OpenGL is a standard that defines an API for communicating with a graphics system. Each graphics card comes with an OpenGL implementation, in the form of a graphics driver. OpenGL functionality might be implemented completely differently in different graphics cards. In fact, the meaning of the various raw inputs and outputs to the card (the "wire" protocol) can change from card to card, but as long as you use the right driver with the right card, you get the behavior you expect, as dictated by the OpenGL API standard.

Putting the interface at the level of the API frees graphics card vendors to experiment with different implementations, and to focus on different performance criteria. One card might excel at lighting effects and anti-aliasing while another can draw a higher number of polygons per second.

A standard at the wire (or pin) level might allow you to use one graphics vendor's card with another's OpenGL drivers, but the cost would be limited flexibility in design choices for developers of graphics cards, and therefore fewer opportunities for end-users to find a card that meets desired price/performance characteristics. Besides, why would one *want* to replace a graphics card without replacing graphics drivers? The OpenGL API defines an interface to the whole card/driver system, and replacing part of that system without replacing the other does not make sense.

Now, this design is only successful because it is (in general) very easy to switch to new graphics drivers when switching graphics cards. Drivers are distributed as dynamic or shared libraries, and are interchangeable. No recompiling or relinking is necessary. If a game had to be recompiled in order to switch to a new graphics card, you can be sure that there *would* be a demand for bridges between one vendor's drivers and another vendor's cards.

This part of the analogy holds for HLA too. If federates are to be expected to switch among different RTI implementations as they move from one federation to another, the process had better be easy. And while HLA does not have a horrible record in this area, there is much room for improvement. The whole concept of HLA is based on it being easy to choose the right RTI implementation for the simulation problem at hand, and if HLA does not live up to that promise, users will be forced to look more closely at bridging solutions, despite their limitations.

Lowering the barrier to getting all desired federates running with a common RTI environment is a problem with both technical and programmatic/business model aspects. Both aspects must be addressed to achieve success.

On the technical side, the important thing is to make sure that, like OpenGL implementations, different RTI implementations are dynamic-link-compatible. If it is necessary to recompile or even re-link a federate to move

to a new RTI implementation, legacy federates that are not under active funding may not ever be able to make the transition. Tool vendors will be forced to maintain separate versions of their tools for each RTI implementation, increasing their cost, and decreasing the rate at which new features can be added. End-users of simulators will be forced to wait for the contractors who built the federates to provide them with new releases for new RTI versions.

By contrast, if different RTI implementations are dynamic-link-compatible, end-users can install and run with a new RTI version as easily as they can upgrade to a new OpenGL-based graphics card - even if the federate is no longer under active development. And third-party tools built using one RTI implementation can be used in a federation using another, without having to wait for a future release.

Given the lack of coordinated effort in this area, HLA has already achieved remarkable progress towards this goal. For several years, end users have been able to switch between the DMSO RTI and the MÄK RTI without recompiling or re-linking. And the latest release of Pitch's pRTI claims the same kind of dynamic-link-compatibility.

On the other hand, ambiguities in the Specification, and other elements of functional compatibility that fall outside the Specification mean that it is sometimes necessary to tune a federate before being confident that it can work with any of several RTIs. For example, certain ill-defined helper functions may need to be avoided, and a "tick" strategy that works for all RTIs might need to be devised. However the recently chartered Dynamic-Link-Compatible HLA API SISO Product Development Group is currently working to resolve the remaining incompatibilities, and to insure that there are no significant *technical* barriers to switching from one compliant RTI implementation to another.

Technical barriers are only half of the problem, however. What about the costs to a federate developer of supporting multiple RTI implementations? If a federate developer has already bought an RTI license from one vendor, can he really be expected to buy another license from another RTI vendor in order to play in a different federation? And what if none of the existing RTI implementations support all of the platforms that are required by a given federation?

I think the answer again lies in realizing that choosing an RTI needs to be a federation-wide decision (as the developers of HLA intended), rather than a federate-by-federate decision.

Federation builder have always had to budget for many elements of the infrastructure that is used by all federates during an exercise: renting or building a dedicated network, purchasing routers and other network equipment, computers, operating systems, compilers, labor for integration, etc. The RTI is just part of the infrastructure necessary to run a federation execution, and in fact, it is one of the least costly elements of infrastructure.

RTIs have never been free. It's just that for most of HLA's history, their cost has been borne by a central organization – DMSO – as opposed to by each simulation program. But as new programs are awarded, it stands to reason that line items will be included for RTIs, just as they currently are for all of the other elements of infrastructure.

Developers of federates will not necessarily be expected to fund the purchase of RTI licenses any more than they are currently expected to bear the costs of renting of the exercise's network. And if they are, the cost of the RTI licenses needs to be factored into the amount of their contracts. Because of natural competition in the RTI market, the cost of an RTI license for a federate typically is equivalent to a few person-days of labor - negligible compared to the costs of building, testing, and integrating the federate into the federation.

What really matters to a simulation program manager, however, is how the cost of an RTI compares to the cost of the other option – employing a bridge, so that federates can each continue to use whatever RTI implementation they have already purchased. Interoperability among different RTI implementations is a fact of HLA life, and one way or another the barrier must be overcome. The only real question is whether using an RTI-to-RTI bridge is a cheaper/better solution than making sure all federates can use a single RTI implementation.

We believe that in almost all cases, using an RTI-to-RTI bridge will be *more* costly than the alternative. Just the costs of *building* an RTI-to-RTI bridge would be many hundreds of thousands of dollars at a minimum. A program-wide RTI license for a federation is usually just a fraction of that. Integrating multiple RTIs and federates with the bridge adds significantly to the cost of that choice, and what you end up with is a solution that has some severe limitations, and that cannot take full advantage of HLA. By contrast, if RTI vendors can solve the technical problem of making it easy to swap RTIs, getting all federates to agree on a common RTI should be a low-cost, high-performance choice.

Finally, what about the situation where no single RTI implementation supports all platforms required by a

federation? In almost all cases, either porting the federate, or porting the RTI (whichever is cheaper) will be less costly than using an RTI-to-RTI bridge. There are at least three commercial RTI implementations that have been ported to 5 or more platforms, and the cost of a custom port is certainly lower than the cost of building and maintaining a bridge.

5. Conclusion

We have described several approaches to overcoming the interoperability barrier caused by in inability for different RTI implementations to communicate directly.

Eliminating the barrier through a wire standard for intra-RTI communication is a tempting idea, but would lock federation developers into a single one-size-fits-all answer for many design choices. It's not that wire protocols are a bad idea; it's just that they are inconsistent with the concept of HLA.

Working *around* the barrier using RTI-to-RTI Bridges is feasible, at least for much of the RTI's functionality. However, bridges are really best suited for applications where thinking of the two sides of the bridge as two distinct federates is a desirable goal. Using a bridge to approximate a single larger federation by linking together several smaller ones does not achieve the desired results. Performance and functionality are both limited.

Finally, *lowering* the interoperability barrier imposed by the existence of multiple RTI implementations seems like the most promising solution. Through a combination of more precise standards and cooperation among RTI vendors, federate developers, and federation program managers, we can make it quite easy for all federates in a federation to agree on a common Run-Time Infrastructure.

6. References

[1] Wood, D.D.: "Implementation of DDM in the MÄK High Performance RTI", *Proceedings of the 2002 Spring Simulation Interoperability Workshop*, Orlando, FL, March 2002.

[2] Post by Nathan Barclay, Teledyne Brown Engineering, to the SIW-RTI reflector, www.sisostds.org, October 2, 2002.

[3] Briggs, K.: "A Required RTI Gateway Standard As a Solution To RTI Interoperability", *Proceedings of the 1998 Spring Simulation Interoperability Workshop*, Orlando FL, March 1998.

[4] Braudaway, W., Little, R.: "The High Level Architecture's Bridge Federate", *Proceedings of the 1997 Fall Simulation Interoperability Workshop*, Orlando FL, September 1997.

[5] Bouwens, C., Hurrell, D., Shen, D.: "Implementing Ownership Management Services With a Bridge Federate", *Proceedings of the 1998 Spring Simulation Interoperability Workshop*, Orlando FL, March 1998.

7. Author Biography

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 Real-Time RTI during its initial development. He has worked on the VR-Link HLA/DIS toolkit and other MÄK simulation and visualization products for over 9 years, serving 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.