Toward Predictive Models of Federation Performance: Essential Instrumentation

Stephen R. Kolek
Steven B. Boswell
Harry M. Wolfson
MIT Lincoln Laboratory
244 Wood Street
Lexington, MA 02140-9185
781-981-4170
{kolek | boswell | harrywolfson}@LL.mit.edu

Keywords:
Federation Performance, Federation Planning, Federation Testing, RTI Performance, RTI Wrapper, FEPW, FMT, FVT, DCT

ABSTRACT: *With support from DMSO, MIT Lincoln Laboratory is developing tools and techniques for measuring and characterizing the performance of HLA federations operating with various RTIs. These tools and techniques are intended for use by federation developers to study, understand, and enhance the performance of their federations. They are intended for application across many types of federations, from time-managed, analysis-oriented federations interested in maximizing the number of executions possible within a fixed time interval to hardware-in-the-loop federations interested in minimizing end-to-end latency.*

An essential initial step in this effort is the development of instrumentation to capture interactions between the federates and the RTI, as well as data flows among the federates. This paper will describe a generic RTI "wrapper" that can capture call patterns (federate-to-RTI and RTI-to-federate) to generate statistical descriptions of service invocations, including histograms of time patterns in these invocations, etc. This instrumentation software will begin to address performance measures and concepts discussed in previous SIW papers by various members of the M&S community with distinctly different concepts of which aspects of overall federation performance are most critical.

In subsequent phases of this effort, data from federations who agree to use these tools and to share their collected performance data will be used to develop predictive models of federation performance. As they mature, these models can be used in conjunction with other tools and test federates for federation planning, development, testing, diagnosis, troubleshooting, and optimization.

1. Motivation

Over the past few years a number of tools have been developed for managing and verifying the construction and performance of HLA Federations. The common purpose of the tools is to reduce the cost and uncertainty in federation development. It is clear, though, that more work needs to be done to assist the federation development process.

Shortcomings lie in two areas. First, individual tools address only a portion of the problem: real time performance monitoring and post-execution analysis.

This work was sponsored by the Department of the Air Force under Air Force Contract F-19628-95-C-0002. Opinions, interpretations, conclusions, and recommendations are those of the authors and are not necessarily endorsed by the United States Air Force.

The full set of these tools offers no practical or direct predictive capability. Users are offered only the hope that if a new Federation under construction is "similar enough" to one which has already been documented it too may work, inspiring far too little confidence in the tools. Second, the tools which gather the most detailed information about a federation appear to be the most demanding to use. Only a few Federation Execution Planner's Workbooks, or FEPWs, have been completed as of the date of this writing.

Several existing tools have been described from the developer's as well as the user's perspective in recent SIW papers [1], [2], [3], [4], [5] and [6]. Section 1.1 of this paper provides a brief review of those tools.

Section 1.2 overviews the programmatics of a new DMSO tool building effort which hopes to draw upon the HLA community's collective experience to incorporate lessons learned from existing tool concepts, techniques, and implementation examples.

Section 2 outlines our technical approach. One of the cornerstones of the approach is the principle that a user's perceived value of HLA design, development, and debugging tools must far outweigh the cost of employing those tools. The new tools must directly and efficiently address federation performance goals, offering both a reliable predictive capability and a real-time monitoring capability, with minimum effort on the part of users and minimum impact on the federation's behavior. Section 3 briefly outlines the data collection tools being developed.

Section 4 details the design of an initial product, an instrumentation tool called the RTI Wrapper. The RTI Wrapper provides the ability to collect source data that can be used to support the development of predictive models of federation performance, and to support federate optimization.

Section 5 briefly discusses our ultimate design goal, a well-researched and well-documented collection of predictive models that can assist federation developers in all phases of the development process. A federation developer would have the ability to access a history of development experience obtained from federations sharing similar characteristics and performance priorities. The developer would be able to use the available models to estimate federation capabilities and anticipate advantageous configurations for the federation. As development proceeded, the models could be applied repeatedly in conjunction with more complete network descriptions and with instrumented pilot trials of the federation, to refine earlier performance predictions and optimize the federation plan.

1.1 Current State of HLA Tools

DMSO has supported the development of a set of tools that are intended to assist in the design and execution of a Federation. Some of these tools include the Federation Execution Planners Workbook (FEPW) [1], Federation Verification Tool (FVT) [2], Federation Management Tool (FMT) [3] and the Data Collection Tool (DCT) [4].

The FEPW [1] can be thought of as a set of forms that are filled out with details of how a Federation will be constructed and what is expected to transpire during its execution. Its value is in defining a "contract of execution" [2] which allows a Federation designer to generate a detailed specification of the Federation Execution.

The Federation Verification Tool (FVT) [2] reads the FEPW, along with additional descriptive data sets generated by other federation planning tools such as the Object Model Development Tool (OMDT). Prior to execution, the FVT checks for consistency in the "contract of execution" defined by these files. During the execution, the FVT acts as another federate (it joins the federation and subscribes to relevant classes, including MOM classes) and keeps track as each individual federate does what it is supposed to do, as defined by these files. During the execution, the FVT GUI displays confirmation of these actions, and after the execution has completed, the FVT GUI can be reviewed to verify compliance or non-compliance of each federate with its "contract of execution."

The Federation Management Tool (FMT) [3] is a federate with a sophisticated GUI front end that subscribes to MOM classes in order to monitor the actions and behaviors of all federates during the execution of a federation. It displays live information during the execution, such as the federation name, time and status, details about individual federates including number of objects, data that is subscribed / published, quantity of data sent or received, etc. The FMT can also provide some control over the Execution, such as scheduling save and restore operations, advancing time, requesting updates, etc.

The Data Collection Tool (DCT) [4] is a sophisticated logger application that records all, or a subset, of the interactions and object updates that are exchanged during an execution.

All of these tools provide value and utility within a well defined realm of federation development and execution. For example, Prochnow and Seidel [5] give a thorough account of the use of the FMT, the FVT, and the DCT, in conjunction with network

analysis tools, statistical techniques, and other instrumentation, to investigate and improve performance of an Analysis federation.

However, existing tools are very limited in their ability to assist in *predicting* how well a given federation will perform; or what computational and networks resources will be required to support the exercise; or what the impact on performance will be if changes are made in the hardware infrastructure that supports the exercise; or what the impact on performance will be if changes are made in the number of federates in the exercise; or the impact if the amount of data generated by the federates changes. The FEPW asks the federation designer to define the computational requirements, or suggests that the designer "guess" the answers, but it does not provide a way to predict accurately what will be required. The FVT assists in verifying if the "contract of execution" data sets accurately reflect what the federates actually do during an execution, but it does not provide any way to determine if limitations in the available hardware or network are restricting successful completion of the exercise. The FMT and the DCT monitor and record several aspects of the data that are exchanged among the federates, but they do not do so in enough detail to generate statistically accurate predictive models of performance. Therefore, the goals of this program will require a new tool set to provide sufficiently detailed information about the federates, network, and computational resources to allow for accurate predictive models.

1.2 New HLA Tools: the Goal and the Process

The primary objective of this new DMSO effort is reducing the cost of building federations, including effort, infrastructure cost, and lead time. We believe that this objective can be achieved with new tools that (1) yield reliable performance predictions, (2) allow observation of federation performance at intermediate stages of construction as well as at completion, and (3) incorporate experience gained with new federates to continue to improve tool effectiveness.

The initial goal of this effort is to build, borrow, and/or adapt useful tools for measuring the performance of a wide range of federations and to provide these tools to federations for collecting data (which they agree to share with this "new tools" consortium). The long-term goal is the development of predictive models for several aspects of federation performance based on the data collected. We expect to iterate this process until we have a robust set of tools that can be used for federation design, development, and performance optimization.

Membership in the DMSO new tools consortium is extended to developers and users throughout the HLA community who are willing to participate and share in tool development, data collection, and model development. At the time of this writing the consortium includes representatives from CMU/SEI, DMSO, Georgia Tech, GTRI, JHU/APL, MIT/LL, Mitre, SAIC, and VTC. MIT/LL has been tasked as the technical lead responsible for new tool development.

Also at the time of this writing four federations have signed on to provide data for this effort. They are the Joint Training Confederation (JTC), Pegasus (JFCOM), Joint Theater-Level Simulation (JTLS), and the Special Operations Forces (SOF) Testbed.

These federations have agreed to use the Federation Performance Monitoring Tools and Federation Performance Models and report on the results to the consortium and the broader M&S community at the semi-annual Simulation Interoperability Workshops. The consortium at large will continue to use these Workshops as the primary forum for sharing tools, models, and results.

2. New Tools

This section outlines the aspects of performance, both qualitative and quantitative, that the new tools will be designed to address.

2.1 Qualitative Performance Goals

HLA Federations can be made to support diverse sets of user requirements including

- time-managed, fast-as-possible
- time-managed, human-in-the-loop
- wall clock time (time "happens"), and
- minimum latency, hardware-in-the-loop.

This flexibility is achieved through the selection of simulations to be used, through the distribution of functionality across different federates and sites, and through the selection and tuning of RTI services to support the federation.

The size and complexity of many federation applications demand the use of resources far exceeding the capability of individual hosts. Simulation tasks are therefore distributed across a network. The mapping of tasks to hosts is implemented with an eye to at least four

considerations: functionality, modularity, host performance, and network segment capacities. Note that in modeling federation performance, we are NOT attempting to address a federation's "fitness for intended use" or any other VV&A-type questions.

As has long been understood, simple broadcasting of messages among a network of federates can quickly swamp the receive processing capability of individual hosts connected to that network. Therefore, one of the fundamental services offered by the RTI is the filtering and forwarding of messages only to where they are needed. The RTI offers other fundamental services such as time management and reliable transport.

While all of these services provide direct gains in federation performance, each contributes in some measure to the processing load faced by each host in the federation. To achieve our goal of predicting and monitoring federation performance, the salient quantitative interdependencies among federation tasks, RTI services, and network resources must be modeled.

2.2 Quantifying Performance

Two questions underlie each of the user-requirements listed in Section 2.1 above:

> Q1: What amount of useful processing can the federation complete per unit time?
>
> Q2: Can processing and exchange of update and interaction messages be completed within acceptable latency limits?

The answers to these questions will depend on the processing capability of each federate host and the aggregate amount of processing that each host is being tasked with, both explicitly, through tasking by the federate, and implicitly, as tasked by the Local RTI Component (LRC). In this paper we intend the term "LRC" to refer to that portion of a specific RTI implementation (if any) that competes with the federate for CPU and memory resources on the local host.

A related question often asked by federation builders is, "how much memory do I need in my hosts to achieve federation performance goals?" This question can be cast in the context of questions Q1 and Q2 above. Each host's CPU/architecture processing capacity and total amount of random access memory available are limited. If the total task load presented to the host exceeds either memory or processing capacity, then latency and loss occur.

We can write the constraint equation as

$$\left[\begin{matrix}cycles\ /\ sec \\ memory\end{matrix}\right]_{\substack{\text{OS, apps} \\ \text{overhead}}} + \left[\begin{matrix}cycles\ /\ sec \\ memory\end{matrix}\right]_{\substack{\text{Federate} \\ \text{process}}} + \left[\begin{matrix}cycles\ /\ sec \\ memory\end{matrix}\right]_{\substack{\text{LRC} \\ \text{process}}} \leq \left[\begin{matrix}cycles\ /\ sec \\ memory\end{matrix}\right]_{\text{Host total}}$$

For example, if host CPU performance (measured in cycles/sec) is exceeded, object attribute updates, as well as LRC transport operations, lag further and further behind federation real-time demands. Likewise, if the task load exceeds the available physical random access memory, the OS begins to swap tasks in and out of slower secondary memory, resulting in increasing delay in attribute updates and LRC transport operations vis-a-vis federation time.

As host processing falls further and further behind, the LRC may be storing ever greater numbers of pending attributes in its buffers, further exacerbating the memory component of this equation. At some point, updates will be generated or acted upon with such latency with respect to federation time that they are useless. Updates can be completely lost if the LRC fails sufficiently to keep pace with the rate of incoming updates. Qualitatively, when things begin to go bad, they tend to go really bad very quickly.

One of the two central tasks of this new tools initiative will be to develop approximations to the functions represented by each of the terms above. The second central task will be to develop tools that observe federate and RTI exchanges, as well as some aspects of host OS behavior (memory use for example), and use those observations to estimate the parameters of approximating terms.

2.3 System Modeling

Federation Performance Models will be developed in terms of three major components: a Federate behavior model, an RTI behavior model, and a communication network model.

We believe that it is completely impractical to try to develop models that represent the internal behavior of federates. Internal behavior is simply too complex and too specific to each application. However, we believe that it is feasible to characterize federates in terms of externally observable statistical representations of resource utilization, and in terms of representations of the messages that the federates emit and receive via their Local RTI Components.

Some pieces of these models already exist in the accumulated products of the HLA working community. Other portions must be developed.

2.3.1 Federate

The complexity of a federation forces us to adopt a probabilistic or statistical approach in characterizing its throughput, latency, and use of computer resources. Each federate can be thought of as a random message generator. In most cases, we do not expect to address the algorithmic complexities behind the generation of any particular type of message (say, a tank turret attribute update). Rather, we are concerned with how to characterize the "random" emission of messages statistically. This statistical characterization boils down to estimating the aggregate probability distribution of messages across several dimensions including message size, time of emission, and destination.

Furthermore, the RTI's filtering and forwarding process, driven by subscription and publication declarations, reshapes the message distributions offered by the federates. These RTI-reshaped distributions characterize the source load finally offered to the underlying communications network.

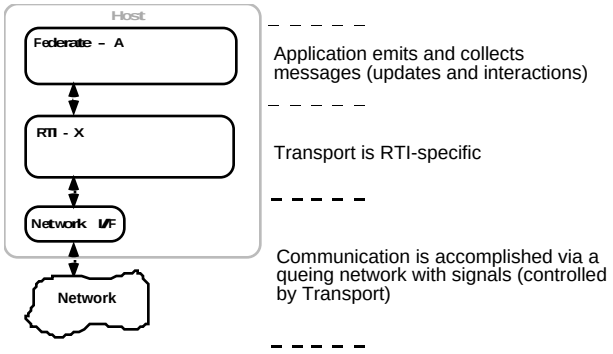

Figure 1. Simple System Diagram.

We can also measure the processing load and memory used in modeling simulation objects and in transmitting attribute updates and interactions. We can use these measurements to develop statistical characterizations of federate performance. Processing load may be inferred from system clock checks. The RTI Wrapper, discussed below, provides a means of sampling the system clock during service calls on the RTI and during callbacks to the federate. The Wrapper can delineate between usage by the federate and usage by the LRC. The Wrapper (or other instrumentation techniques) can also infer memory use during execution by making queries to the operating system. In some cases, it will be possible to separate out the LRC contributions to memory use and processing by using a very simple deterministic LRC as described in [7].

We can then use statistical methods to model performance for a particular federate or type of federate, given the measurements described above as well as other characteristics (implementation language, platform, use of time management, etc.). Common techniques for analysis would include linear models (*e.g.*, linear regression) or queuing models. Some modeling may be reasonably complex and may vary significantly from federate to federate. For example, processing load may be strongly nonlinear as a function of the number of objects simulated by a federate or federation. Other quantities, such as memory used per object class instance, may require no modeling, rather, just a mean or constant value. It is likely that expert knowledge will be necessary to partition models for a given federate based on federate state (*e.g.*, a SAF tank platoon will emit far more interactions during an encounter with enemy than when traveling with friendly forces) as demonstrated in [8].

With the help of some of the consortium test federates mentioned earlier in this paper, we plan to develop tools that aid in these types of analyses.

2.3.2 LRC

The RTI plays two fundamental roles in the federation. First, it offers federation-wide coordinated management services such as time synchronization and guaranteed-delivery message service for certain types of essential information. Second, it significantly reduces the amount of host processing required to maintain the state of remote objects by ensuring that the host has to deal only with objects and interactions that are highly likely to be relevant to its simulated objects.

However, in achieving these benefits, the host trades off a portion of its random access memory and some amount of its CPU processing capacity. In Figure 2 a conceptual RTI is depicted with notional subcomponents. Some of the major consumers of memory are shadowed in gray.

Some RTI components explicitly trade message delay for the ability to synchronize actions across the federation (Time Manager), reduce network traffic and therefore reduce individual host receive processing (Bundler), or guarantee the delivery of certain types of messages (Reliable Distributor). Some of the components that can impose substantive delay are shadowed in black in Figure 2.

Recall that we distinguish the Local RTI component, or LRC, as that portion of an RTI implementation which competes with a federate for host CPU and memory resources. RTI implementations will vary significantly. The component breakdown shown in Figure 2 is intended to be illustrative, not definitive. Particular RTI implementations may not include (or may include only trivial cases of) some components mentioned in this section.

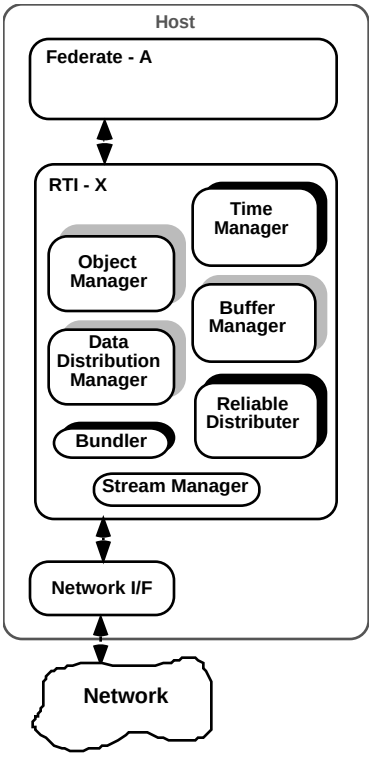

Figure 2. RTI components with first order memory consumers shadowed in gray and first order delay imposers shadowed in black.

In order to model the RTI processing task costs and their effects on throughput and latency throughout the federation, it will be necessary to create functional models for processing and memory use of each of the key components of an RTI. Furthermore, we need to develop a quantitative understanding of the interactions among these components.

3. Data Collection Tools

The first key tool needed for gaining insights into the interactions of federates with the RTI is a generic interface layer that we will call the "RTI Wrapper."

The RTI Wrapper sits between the federate and the RTI and monitors transactions between the two; see Figure 3. From this vantage point, statistics can be gathered on time, size, type, destination, and other transaction characteristics that may be of importance in analyzing and modeling the behavior of federates and RTIs.

The RTI Wrapper will conform to IEEE 1516 and should thereby effectively conceal its existence from the federate. Furthermore, use of the RTI Wrapper should require little to no extra effort on the part of a federation developer. A design which achieves both of these goals is outlined in Section 4.

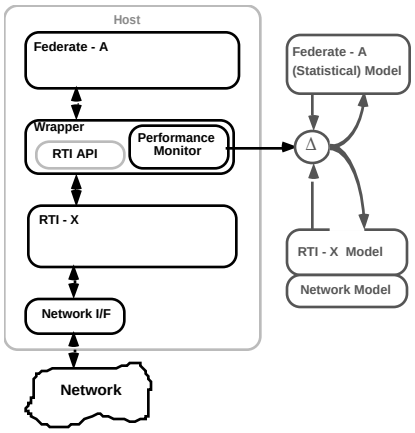

Figure 3. The RTI Wrapper collects statistics on transactions between the federate and LRC and on host processing performance and memory use.

The second tool we are developing is one we've called the "Network Sounder." Network layout changes may occur quite often, especially in the early stages of the development of a new, or the reconstruction of a well-understood, federation. While the Federation Execution Planner's Workbook (FEPW) provides useful information on the detailed objectives of the federation, timely answers to very practical implementation questions serve to help Planners construct and debug their federations. These same answers are needed to predict the expected performance of the federation in its current state (whether that state yet matches the full expectations of the federation planner or not).

The Network Sounder, shown in Figure 4, builds a federation connectivity model and forwards it to federation modeling hosts. It may be initialized from the federation host table in the FEPW or some other source. The Sounder uses a variety of network connection methods to provide immediate answers to the questions: Are all host computers that will be used to support the federation available, and are they

where they are supposed to be (IP address, host name, etc.)? Are the host computers able to reach each other over the network and support all the types of network connections that are expected (*e.g.*, TCP, UDP, Multicast, etc)? What other services and/or hosts are sharing the network, and are they putting unanticipated or excessive loads onto the network? The Sounder may be executed from a host sharing connectivity with, but not necessarily a member of, the federation.

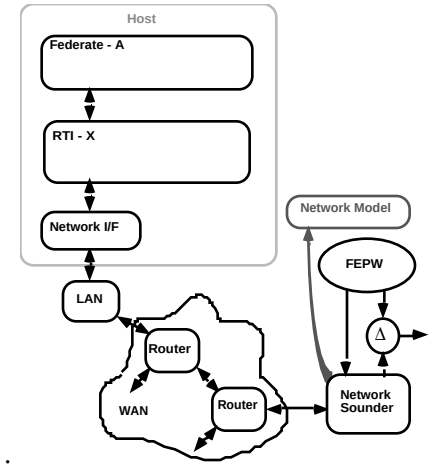

Figure 4. The Network Sounder determines the actual current network layout and may compare it with a FEPW host table for execution debugging.

Among network types currently supported by RTIs, IP networks are employed in the most widespread and disparate configurations and will require the most complex Sounder design. Thus our initial Sounder development efforts will be directed at IP networks. Transfer of Sounder functionality to other communication infrastructures will occur in later phases of the tools initiative.

4. RTI Wrapper Design

As indicated in Figure 3, the RTI Wrapper is a mediating layer that can be inserted between any HLA-compliant LRC and any HLA-compliant federate for the purpose of gathering information. The Wrapper has only public access to the services of the RTI Ambassador, and likewise to the Federate Ambassador.

The first implementation of the RTI Wrapper will be in C++. An initial design for the Wrapper, current at the time of writing, is given here. It should be noted at the outset that the design has been partly but not completely verified, and is subject to revision as implementation proceeds, particularly as new ideas are developed by the user community about the process and techniques of modeling federation performance.

The RTI Wrapper mimics the C++ definition of the RTI class, as specified in Annex B of IEEE 1516. The interface that the RTI Wrapper presents to a federate is indicated in the code fragment below.

```
class RTIWRAPPER : public RTI {
public:
   class  FederateAmbassador;

   class  RTIambassador : public RTI::RTIambassador {
   public:
      #include "RTIambServices.hh"

      // Wrapper Control API
      // (notional)
      void monitor();
      void add();
      void remove();
    };
};
```

The Wrapper interacts with a federate and LRC similarly to the distributed Logger tool that is described in [9]. To use the Wrapper, a federate simply creates an instance of the class RTIWRAPPER::RTIambassador, rather than the IEEE 1516 equivalent, RTI::RTIambassador, from which the Wrapper inherits. Subsequently, as regards use of HLA services, the presence of the Wrapper layer is transparent to the federate. The class RTIWRAPPER::RTIambassador instantiates the actual RTI Ambassador, to which it relays service calls invoked by the federate. When the federate joins a federation, the Wrapper creates an instance of the RTIWRAPPER::FederateAmbassador class, and passes a reference to that class instance in the joinFederationExecution call on the actual RTI. The RTIWRAPPER::FederateAmbassador class retains and subsequently invokes callbacks via the Federate Ambassador reference that was originally provided by the federate.

Besides transparency to the federate, design goals for the RTI Wrapper are that it be

- lightweight: The Wrapper should impose minimal overhead apart from intrinsic analysis and data collection costs.

- extensible: The Wrapper is a support tool in an ongoing process of developing performance models and diagnostic capabilities for a wide variety of purposes. The wrapper should

readily incorporate new collection regimens and new analysis procedures as this process evolves.

- adaptive: The RTI Wrapper will be supplied with a default behavior for fully transparent use. It should also be convenient to specify different instrumentation configurations for different runs; that is, to specify choices as to what data is collected, what analysis procedures are to be invoked, and how results are to be reported. Ideally, it is desirable to be able to turn on and turn off specific capabilities dynamically during an execution.

The initial design for the RTI Wrapper is to have it implement each of the RTI Ambassador and Federate Ambassador services using the same architecture and procedural framework. As indicated in Figure 5, the execution flow in the Wrapper may be conceived of as passing through three stages, namely a stage prior to the invocation of the actual RTI service, the performance of the service by the actual RTI, and an interval between return of the actual RTI and return to the federate by the Wrapper. The Wrapper has an opportunity to make measurements in the first and third of these stages. Analogous staging occurs in the Federate Ambassador.

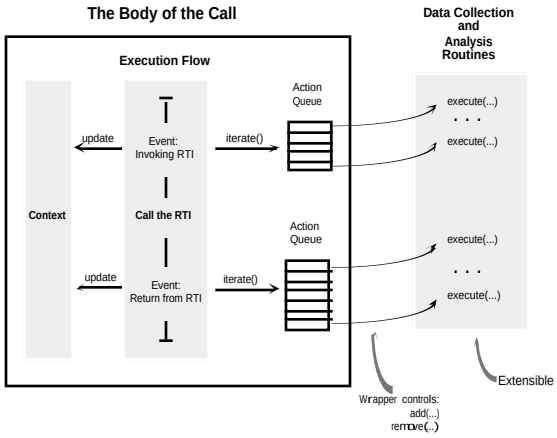

Figure 5. Wrapper implementation of a generic service call on the RTI Ambassador.

In Figure 5 the stages are labeled as "Events", though they are not asynchronous or interrupt-driven as are, say, events of the Java AWT. Other occasions exist where the Wrapper might be intended to execute particular instrumentation functions (for example, an occasion might be the receipt of an exception, or expiration of a timer). As federation performance models are developed, needs may be identified for analysis or data collection in response to such events. Therefore, the Wrapper implementation of an RTI Ambassador service is designed, notionally, as a set of queues, one for each event class that is identified in the federation modeling process. The queues are occupied by references to event handlers, each of which is implemented to perform a particular instrumentation function. The event handlers all inherent from a base event handler class. The base class supplies an execute(..) method, which performs the desired instrumentation function, and it supplies a common means of providing arguments to the execute(..) method.

Some event handlers may provide context information for subsequent use by other instrumentation functions. Examples of such context are:

- a static counter of method invocations
- (static) time of last invocation, for calculating time between invocations
- clock time immediately prior to calling the actual RTI method, used upon return to calculate execution time for the method.

The event handler class will have methods to obtain context information from the Wrapper API method which is invoking it.

The executable code in the Wrapper API implementation simply iterates over the event queue for each "event" as the event becomes active, calling each handler in the queue in turn. If there are no instrumentation functions registered for a particular API call, the Wrapper is a direct pass-through to the matching Ambassador call.

The control portion of the RTI Wrapper utilizes an add(..) method to register event handlers with the appropriate event queues of individual Wrapper API services. A remove(..) method performs the converse. The add(..) and remove(..) methods may be called at any time and can be used to configure the instrumentation behavior of the Wrapper adaptively during a federation execution.

The use of callback queues and event handler classes is intended to decouple the details of instrumentation functionality from the execution framework of the Wrapper, and to make the development of instrumentation capabilities as open-ended as possible. To develop a new capability, it is necessary only to implement a derived event handler class and add it to the library of such classes; no

modifications to the RTI Wrapper proper would be required.

Maximum extensibility occurs if federate and federation developers can supply custom instrumentation functions and request the wrapper layer to invoke these functions on behalf of the developers. Constraints on customization may exist because of possible contention with functionality that is required of the Wrapper layer (for example, interference with precision timing information). However, we believe that it should be possible to provide a useful customization capability to the user community by doing two things:

- documenting the characteristics of the event handler base class, including protocols and other assumptions governing the relationship with Wrapper resources, and
- promoting the add(..) and remove(..) methods used within the Wrapper to activate or deactivate specific event handlers to the public interface of the Wrapper.

More information about the RTI Wrapper and its use should be available at the Fall SIW and in future documentation.

5. Deliverables, Federation Catalog

Deliverables from our efforts will include:

- Predictive modeling tools and manual.
- Prototype predictive models for specific RTIs. (We will use and compare RTI NG and RTI 1.3 to show universality of our approach.)
- Prediction and measurement comparisons from specific federations with help from consortium members.

Performance models with good predictive capability are needed at all phases of the federation development process, from initial sketches of a proposed federation, to management of a mature federation in execution. Also, models are needed that will serve a wide variety of federation types and operating environments.

We anticipate the development of a new resource we are calling the Federation Performance Model Library. This library would contain a collection of prototype predictive models. Each model would employ configuration variables that have been identified as predictive for certain types of federation, certain infrastructures, or certain analytical concerns (*e.g.*, balancing latency against throughput in one case, concentrating on Data Distribution effectiveness in another). Each model will be quantified with coefficients for the predictive variables derived from instrumentation such as the RTI Wrapper, as well as from other available sources. The Federation Performance Model Library would aid in the construction of federations by giving effective access to the experience of prior federation development efforts that share related traits and concerns.

The Model Library would be automatically extended as new federates are modeled and analyzed via the RTI Wrapper. An official DMSO Federation Performance Model Library could reside at the DMSO website, where new submissions could be submitted for "verification" and "certification" before they are accepted for wide scale distribution to the HLA community.

6. Acknowledgements

The authors wish to thank Duncan Miller for his careful review and his contributions to the ideas in this paper.

7. References

[1] Marnie Salisbury, David Seidel, "Execution Planning for HLA Federations" 99F-SIW-092, Fall Simulation Interoperability Workshop, 1999.

[2] Margaret Loper, David Rosenbaum, "The Federation Verification Tool", 98F-SIW-123, Fall Simulation Interoperability Workshop, 1998.

[3] Mak Technologies, "Federation Management Tool User's Guide", http://fmt.mak.com/.

[4] Paul Perkinson, et. al., "How to Plan and Execute Data Collection and Analysis for HLA Federations", 99S-SIW-176, Spring Simulation Interoperability Workshop, 1999.

[5] David Prochnow, David Seidel, "Optimizing Performance of an Analysis Federation", 00S-SIW-025, Spring Simulation Interoperability Workshop, 2000.

[6] Roger Wuerfel, Jeffery Olszewski, "An RTI Performance Testing Framework", 99F-SIW-127, Spring Simulation Interoperability Workshop, 1999.

[7] Rune Sjöström, Ulf Johansson, Andreas Nyberg, "RTI NG and pRTI Performance for Simulated Aircraft, Real Time Updates" 99S-SIW-058, Spring Simulation Interoperability Workshop, 1999.

[8] Jenifer McCormack, Cristl Weckenmann, Gene Lowe, "Development of a HLA Constructive Performance Model" 99S-SIW-146, Spring Simulation Interoperability Workshop, 1999.

[9] Gerry Magee, Graham Shanks, "Lessons Learned from an Implementation of a Fully Distributed Data Collection Tool, " 99S-SIW-084, Spring Simulation Interoperability Workshop, 1999.

Author Biographies

STEPHEN KOLEK holds a BS in Physics and MSEE in Communication System Engineering from Worcester Polytechnic Institute. Recent contributions in the Distributed Systems Group at MIT/LL include RTI 1.3's bundler and buffer manager, research in electromagnetic compatibility for next generation civilian air traffic control digital radio, and systems engineering on midair collision avoidance for military aviation.

STEVE BOSWELL is a Member of the Technical Staff at MIT Lincoln Laboratory. He has a Ph.D. in Mathematical Sciences from Rice University. He has been involved with Distributed Simulation for five years. He participated in the development of the STOW RTI Prototype and the DMSO RTI v1.3.

HARRY WOLFSON is a Member of the Technical Staff at MIT Lincoln Laboratory. He was responsible for the design and implementation of the Reliable Message Transport Service, and its underlying protocol, for the STOW RTI Prototype and the DMSO RTI v1.3. Currently, Harry splits his time between the RTI, and development of Intrusion Detection Systems. He received an MSEE from the University of Southern California and a BSEE from Northeastern University.