Automated HLA Federation Interoperability Testing Tools[1]

Felix Rodriguez
Miles R. Fidelman
Brian Spaulding
MÄK Technologies
68 Moulton Street
Cambridge, MA 02138
617-876-8085
frodriguez@mak.com, mfidelman@mak.com, bspaulding@mak.

Erik S. Hougland, PhD
Keith Nielsen
NAVAIR Orlando TSD 4617
407-380-4670, 407-380-4432
Erik.Hougland@navy.mil, Keith.Nielsen@navy.mil

Keywords:
Federation, Federation Agreement, FOM, HLA, Interoperability Testing,
Trainer Test Procedures Results Report, TTPPR

ABSTRACT : *Integrating a distributed simulation exercise federation is a complicated, time-consuming effort with much of the work involving interoperability testing of various sorts. Issues involved in integrating an HLA (High Level Architecture) simulation include: ensuring that underlying network connectivity is in place; configuring and bringing up the HLA Run Time Infrastructure (RTI) for the exercise; verifying that all federates are using the same Federation Object Model (FOM) , that all federations are generating and interpreting data in accord with that FOM, and that all. specific requirements, expressed as federation agreements, are met.*

This paper describes a phased research program to create a set of automated HLA interoperability testing tools designed to simplify and speed up the process. The set is based on two primary tools. The first is an RTI monitor that retrieves information about network connectivity and the internal communication process. The second is a FOM monitor that reads the FOM to determine the validity of the data and ensure that all joined federates are following their federation agreements. Additionally, this paper relates experiences in using and evaluating these tools in large federations.

[1] Work described in this paper was partially sponsored by NAVAIR Orlando, the JSF Program, and the US Small Business Administration under SBIR N03-022, Automated Interoperability Testing.

1. Introduction: The Complexity of HLA Exercises.

HLA-based distributed simulations are getting larger and larger. Unthinkable just a few short years ago, some exercises have surpassed tens of thousands of entities, and there is no reason not to expect them to get even bigger in the future.

Millennium Challenge 2002 (MCO2) brought together live field exercises and simulations involving approximately 13,500 people and 35,000 simulated entities, spread across two dozen locations, nationwide. The Joint Experimental Federation (JEF) – which stimulated service C4I networks and provided a common operating picture for the exercise – integrated roughly 90 simulations (AWSIM, JCATS, JSAF, etc.). [1] [2]

Integrating such a large collection of participants, presents a considerable challenge. Making things worse, while the size and complexity of these exercises keep growing, the time available for staging is actually shrinking. Large exercises are becoming more frequent, and some are moving towards an environment where distributed training environments will operate continuously. Programs underway include the Air Force's Distributed Mission Operations (DMO), the Navy Aviation Simulation Master Plan (NASMP), and DARPA's Training Superiority Program (DARWARS). [3]

As we move towards these large and larger continuously operating distributed federations, there will be reduced opportunity to bring down the system while a new federate is integrated. New federates must be designed individually and separate from the full federation. These must then simply be able to plug in to the federation and work with other similarly designed federates.

2. Integration and Interoperability Challenges.

Designing an entity to plug-and-play into a large federation is a very difficult task. Developers have to worry about their usual problems: making sure that the developed entity works as expected within their own environment, but they also have to worry about an entirely new set of problems. Now entities must also follow whatever rules have been set by the federation it wants to connect to, and make sure that it plays nice with all other entities currently connected to it. Additionally, the developer must also have a degree of clairvoyance and make sure that the new entity will play well with existing entities, and with entities that have not been created yet!

The manager of the federation has his own set of new challenges to resolve. First, very clear interoperability specifications and rigorous testing must be written if we are to approach the desired level of plug-and-play. If there is even a smallest discrepancy in unclear specifications, the entire exercise could fail.

Even with the most rigorous of testing schemes, due to the large number of possible entities, a manager must also plan for the worst. Entities connecting to the federate may not be working as expected and may waterfall into other entities behaving erratically. Identifying the exact cause of a problem that may be as small as slight differences in positioning or large enough to take down the entire exercise is a tough task given the amount of variables involved.

A non-exclusive list of interoperability considerations that must be specified and tested includes:

Computing Platform Issues: Is the federate installed on the right hardware and operating system? Is there sufficient CPU power and memory to operate properly under full load?

Network Connectivity: Is there network connectivity from the federate to all other members of the federation? Are names and addresses assigned properly? Are intervening firewalls and routers properly configured to allow traffic to pass? Is there sufficient bandwidth and sufficiently low latency on all paths?

Protocol/Middleware Configuration: If Distributed Interactive Simulation (DIS) is being used, does the federate implement agreed upon extensions? If HLA is in use, does the federate use

the same RTI as the federation, and is it properly configured? If the federate uses another interface (e.g., Link-16, TENA) is it properly bridged? Does the federate interoperate properly with the protocol or middleware in use? Is the middleware properly configured to make use of its underlying network layer?

Data Compatibility: Does the federate use the same object model as the federation and is it implemented properly? Does the federate use the proper conventions for naming new object instances? Is data represented correctly (size, endianness, encodings, ranges, tolerances, etc.)? Does the federate properly implement the same spatial model, coordinate systems, dead reckoning, and update policies as the rest of the federation?

Simulation Assumptions and Behavior: Does the federation implement the same behavioral assumptions as other members of the federation?[2]

Time Management: Does the federate conform to the federations time management scheme?

Data Reporting: Does the federate report data required for system monitoring, after-action review, etc.?

Exercise Management: Does the federate properly implement the federations startup, shutdown, fault management, and fault recovery, schemes?

3. Federation Agreements: Defining interoperability specifications.

When federates are developed individually and separate from each other, interoperability requirements must be identified to a sufficient level of detail that allows all entities to communicate with each other. Under the Department of Defense modeling and simulation environment and HLA, the Federation Development and Execution Process (FEDEP) provides this environment with a well defined architectural and procedural context.

The FEDEP defines a top level procedural model for defining, developing, integrating, testing, and executing federations. A core focus of the process is a detailed "Federation Agreement" that defines and documents all of the rules that participating federates must adhere to.

3.1 What's In Federation Agreements?

Federation agreements are both contractual and technical in nature. From a contractual point of view, federates that participate in a specific federation have a responsibility to comply with its Federation Agreements.

From a technical point of view, Federation Agreements define what it means to be interoperable with a federation – providing the basis for establishing specific interoperability test and acceptance criteria.

The Navy Aviation Simulation Master Plan Federation Agreements Document (FAD)[4] provides a recent, well worked out example of a collection of federation agreements – in this case for Navy aviation training. The plan specifically addresses interoperability with other training systems - including Navy Continuous Training Environment (NCTE), Marine Corps Aviation Simulation Master Plan (MCASMP), Battle Force Tactical Trainer (BFTT), Air Force Distributed Mission Operations (DMO), and Joint Strike Fighter (JSF).

The FAD includes details regarding the use of HLA, the FOM, data representations, federation operations/testing/status monitoring, execution environment (gateways, hosts, common data repositories), characteristics of specific federates, data logging, IP address assignments, and a myriad of additional details.

A companion document, the NASMP Federation Agreement Test Plan (FATP), is being developed that enumerates test requirements, test cases, and

[2] A particularly good example comes from MC02: "AWSIM simulates aircraft only when they are flying. Once they are destroyed they are deleted from the federation. However JCATS and JSAF aircraft fall to earth and remain there as destroyed hulks. This initially confused AWSIM aircraft, which would continue to engage destroyed aircraft and even fly into the ground after them." [2]

test tools such as the Federation Agreement Compliance Test Tool (FACTT).

A Federation Agreement is, for the most part, written in a human readable format. This has the advantage that a programmer designing a federate can easily find what he needs to do to make it work. However, it has the disadvantage that it is not machine readable. There is no easy way to ask a computer to verify that a federate is perfectly complying with its Federation Agreement.

Without specialized testing tools that know exactly what a federation agreement requires, it would be nearly impossible to guarantee that every federate is performing exactly as specified in all but the smallest and simplest of exercises.

4. Interoperability Testing Tools

There are three main areas where testing tools can be made to simplify development and integration.

The first area is instrumentation. This involves providing visibility to the behavior of a federate under test.One type of instrumentation software prints, possibly to a file, all of the raw data it can detect on the network. Two examples of these are DMSO's Federation Verification Tool (FVT) and MÄK's Net Dump.. Both log data and prepare a variety of reports based on the logged data. [6]

The second area is a testing environment. A common tool that supports this is the Simulation Interoperability Test Harness (SITH) [7]. The SITH provides a scripting language that allows simple entities to be emulated in order to make sure that your own developed federate can interact correctly.

The third area for testing are management tools; things that can control and coordinate the simulation. DMSO's Federation Management Tool (FMT) [8] provides top-level monitoring and control over federation execution – it reports such information as federation name, local time, federation time, objects owned by specific federates; it can invoke federation saves and restores, advance time, set reporting details, etc.

Of these, instrumentation is the core capability – as what can be stimulated and measured sets the fundamental limits on all testing. Based on experience developing and supporting HLA products, it has been concluded that a finer grained level of instrumentation is required.

For this purpose, we propose a two-tiered approach. The first is an improvement to the RTIspy software. The RTIspy is an available product that specializes in testing connectivity and data communications within an HLA exercise.

The second solution is a new product called the FOMspy. This tool verifies correctness of data as it is seen inside the exercise.

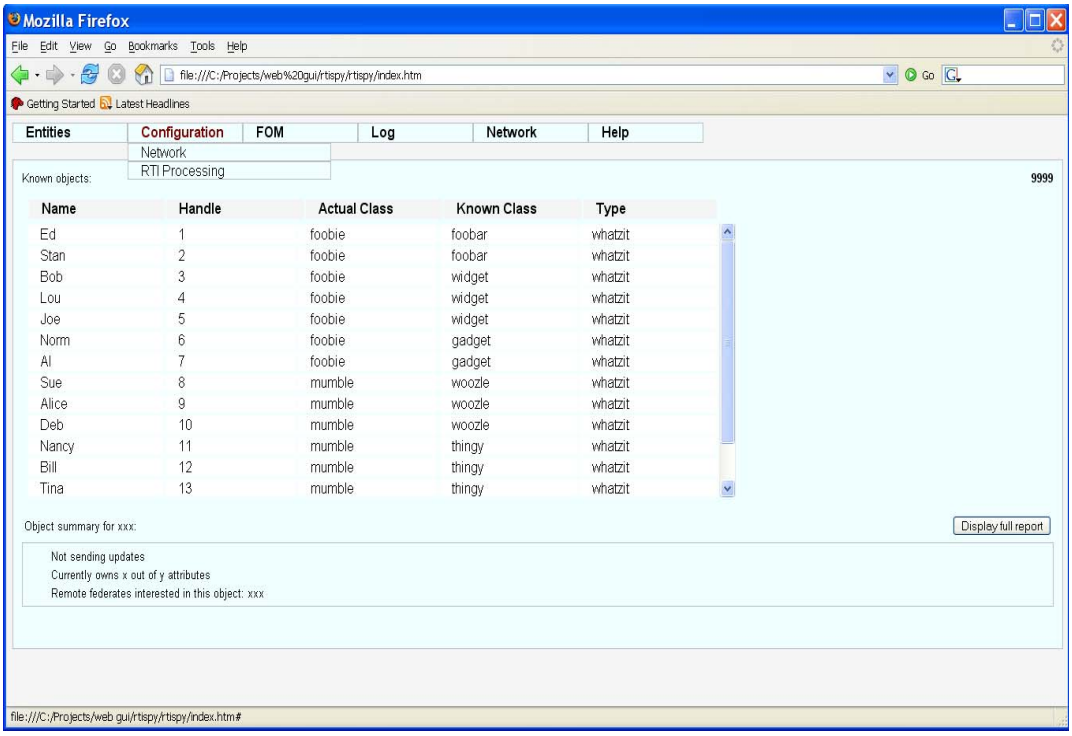
Using both tools, the tester can verify exactly what data is being passed between the federates and that this data is correct. Working in tandem, the tools provide a useful package in the quest for plug-and-play interoperability.

5. Web-based RTIspy

Using traditional techniques, the HLA RTI is a black box – an application program interface (API) call is made and returns, which may result in data and interactions appearing at other RTI interfaces. But, if expected results do not appear, there is little or no information available for diagnosing the problem.

Instrumenting the inside of the RTI allows an API call to be tracked from its inception, through all stages of its execution. By looking inside the black box, it becomes possible to examine:

- whether an API call is made at all (a federate may think it made the call, but it may not be seen by the RTI; the RTI may think it made a callback but it does not get to the application)
- whether an API call is properly formatted
- state changes resulting from API calls
- network transactions resulting from an API call
- incoming network transactions and resulting RTI activity (state changes, callbacks)



Panel A: Entity lists in the WEBspy

latency associated with each step of a function invocation

resource use associated with each step of a function invocation (CPU cycles, memory, network bandwidth)

objects and interactions published and subscribed to by a local RTI component (LRC)

It is important to note that such internal instrumentation of an RTI will be tied to a specific vendor's RTI – and thus is primarily benefit in environments using that vendor's RTI. However, since RTIs are intended to be interchangeable at the interface level, it is conceivable to use one RTI (i.e., a highly instrumented one) in a test federation and a different RTI in an operational federation.

The commercial RTI product used in this work was the MÄK RTI. It has a feature, the MÄK RTIspy, which provides a window into its internal functioning in the form of both an API and software developer's toolkit (allowing programmatic access to the innards of the RTI) and a graphical debugging tool built on top of the toolkit.

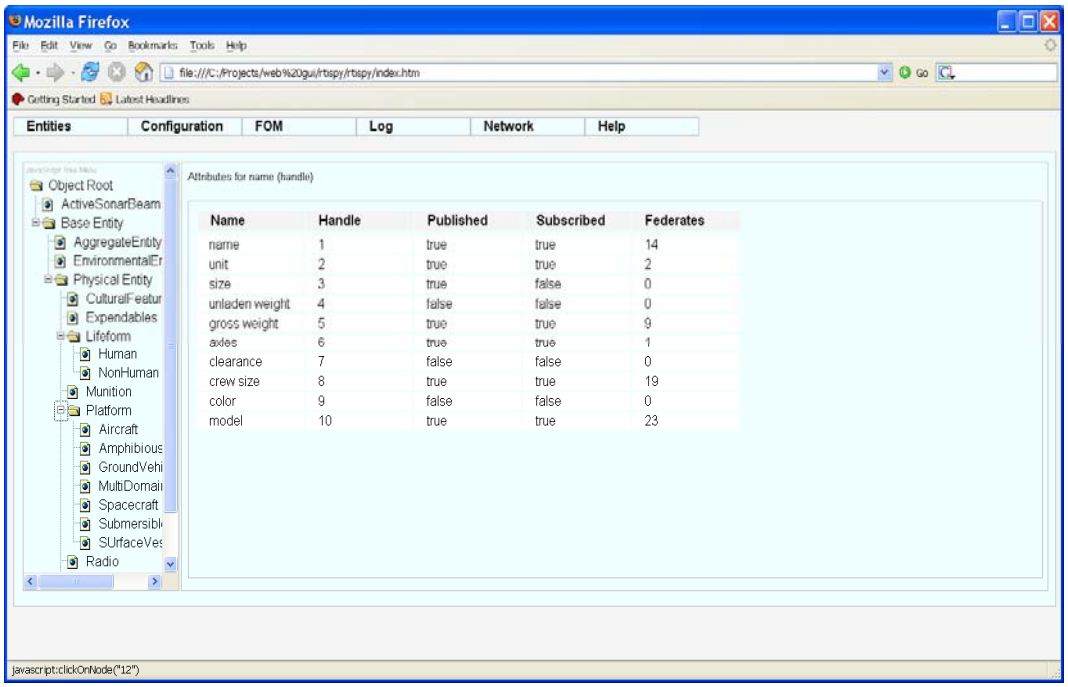
The RTIspy allows federation developers to build plug-ins to alter, extend, and query any aspect of the RTI's functionality. Developers can change the RTI's "wire format" by implementing new network messages, perform encryption or compression on network messages, register callbacks to be executed whenever certain services are invoked, tune performance based on program-specific requirements, and override default implementations of key services. The API has been used to implement shared memory as an alternative to network-based data transfer.

With NAVAIR SBIR sponsorship, the RTIspy is receiving improvements that range from additional traffic analyzing tools to new graphical information.

The biggest change, however, is in the interface itself.

5.1 Web Interface

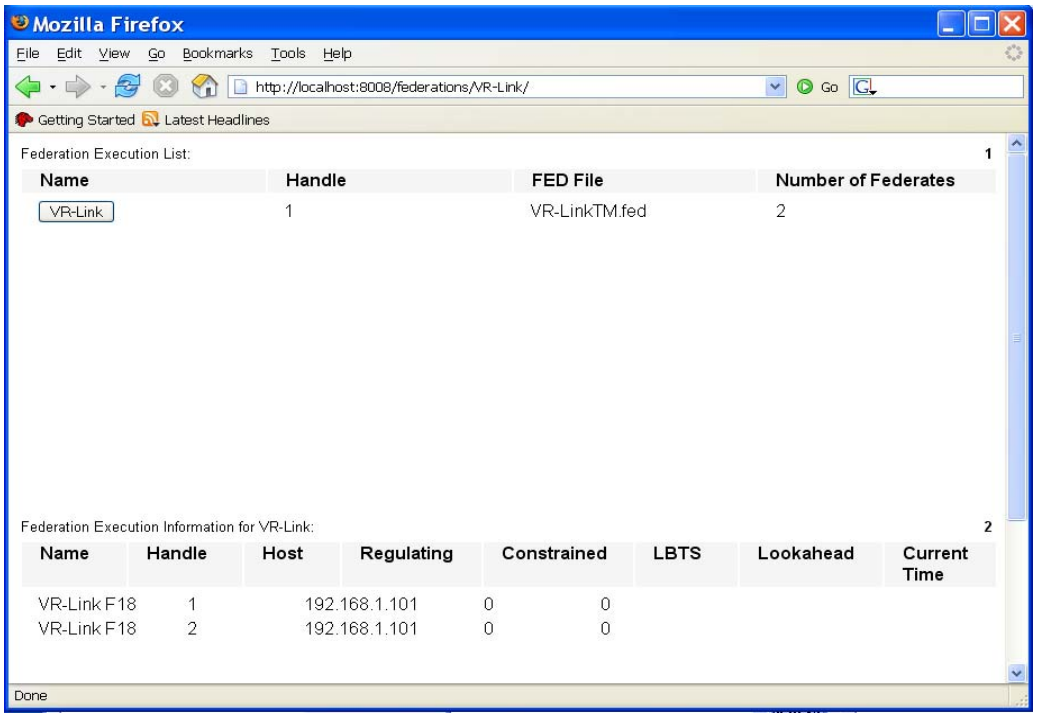
The RTIspy works by attaching to the LRC console of each federate. In order to display the data it has collected, it creates one new window for each LRC console.



Panel B: Object viewer in the WEBspy

This is inconvenient when you want to view the data for multiple federates since you would have to move around to each physical location of the federate to view its own data.

A better solution is to have each federate connecting to a central repository to can grab the data for everything in the network. For this purpose, the SBIR program has created the WEBspy, a web based viewer that assembles data from multiple RTIspy instances. One can use any computer with appropriate access privileges, and use a web browser to point to the WEBspy's location. From there one can view objects much in the same way that you could using the traditional RTIspy GUI. The difference is that one can access all federates instead of just the ones controlled by the computer in front of you.



Panel C: Federation listing in the WEBspy

5.2 Automated Consistency Checking

The second major change involves automation. When large exercises are running, it becomes tedious to check every single federate for possible inconsistencies.

Automation avoids this tedium. If the computer can automatically do most of the testing work for us then there is less of a chance that the tester might miss a critical task.

Reported experience shows that a large number of errors in exercises are due to configuration file inconsistencies between the federates and the RTI. In order to avoid this problem, the SBIR effort includes an automatic configuration file tester.

There are two major aspects to this feature. The first is a RID file tester. The RID file is used by the RTI to determine how the federates are going to communicate. It contains information such as how the sockets are configured, which subset of HLA is going to be enabled, and whether or not reliable communication will be used. A misconfiguration of this file could completely disable a federate from being able to talk with the network.

The RID file tester tries to solve this by sending the set of data that it needs consistent through the network early during the connection procedure. The LRC can take a look at this data and compare it to its own values for the RID file. If there is an inconsistency, then the tester will quickly be able to find it and hopefully eliminate the inconsistency. Alternatively the tester could let the RTI take care of these inconsistencies and override the necessary variables at runtime. Thus guaranteeing that all federates will be compatible.

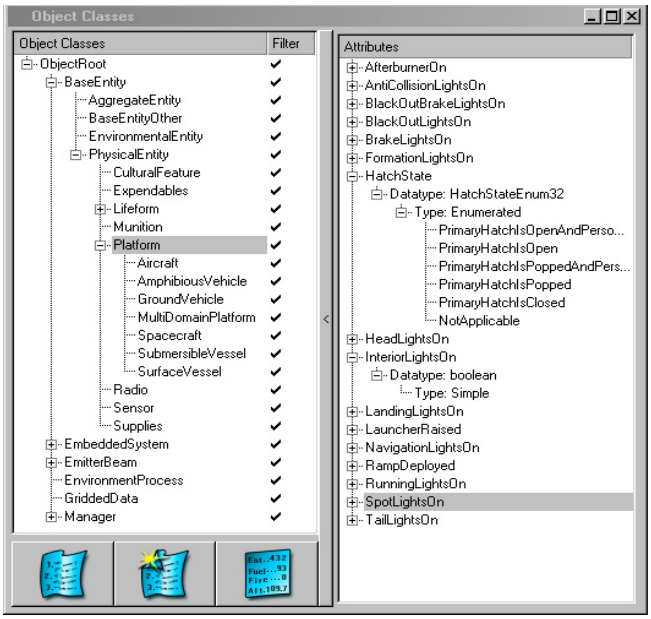
The second configuration file being tested is the FED file. The FED file contains a list of the objects and interactions as well as the attributes available inside each of these. Without this file, a federate would not know the data vocabulary in an exercise.

The MÄK RTI has the ability to send the correct file to all of the federates at the time of connection. The FED file to be used by the federation is defined by the first connected federate. Each subsequent federate joining the federation can then request the common FED file from the RTI, guaranteeing that every object is using exactly the same version.

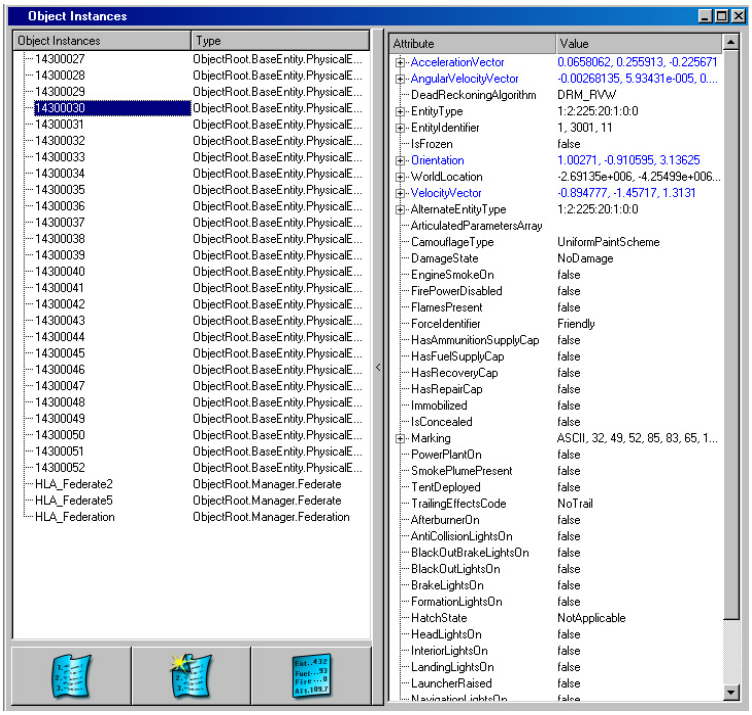
6. FOMspy

The FOMspy, a product of the SBIR effort, is a debugging tool whose main purpose is to verify correctness for all data being passed during runtime in an HLA Exercise. It complements the RTIspy in that it works at a higher level, testing the data it receives without worrying on the RTI activity that caused this data.

As mentioned previously, a Federation Agreement is not machine readable. However, HLA specifies an Object Model Template (OMT) which does have a machine readable structure. The OMT file is an extension of the standard FED file that contains not just information on what attribute each object contains, but also exactly how data is stored in each attribute. For example, a FED file specifies that an “Entity” contains an attribute called "Location". An OMT file also notes that the attribute called Location is actually three double values called X,Y,and Z.



Panel D: The Object Viewer in the FOMspy

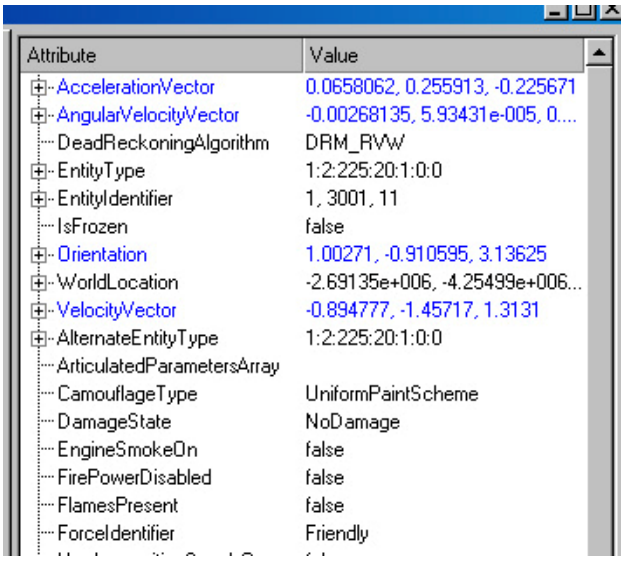


Panel E: The Instance Viewer in the FOMspy

When started, the FOMspy reads all of this OMT data uses this information to create a translator for all binary data received. Its purpose is to display values in a way that is more useful to the user than the raw bits. "Location" described above, would be shown something like this: "3.45, 67.23, 0" as opposed to the byte data "F5E593CA". A character string would translate to the actual string. Booleans translate to "true" or "false". Enumerations get converted to their names instead of the numbered value.

The FOMspy can also handle more complex attributes, such as arrays, variants, and collections, by dividing the items into sub-particles and displaying them in a tree. Only the most complicated and proprietary data formats are not supported out of the box.

Even in this extreme situation however, the user is not stuck when faced with proprietary data. The FOMspy contains a full plug-in system that enables one to add any encoding types not supported out of the box. The user can implement a new C++ class that describes how to translate the raw data to a readable format.



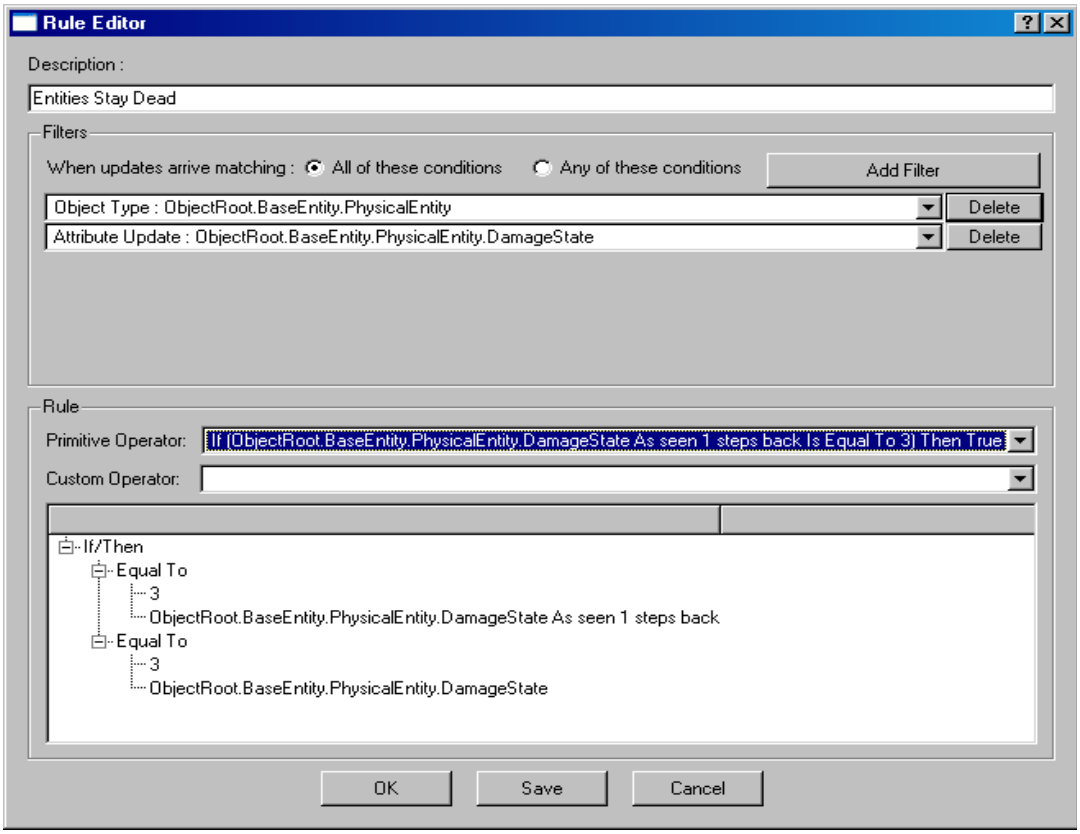
Zoom In on the Instance Viewer

The final result of this effort is that the FOMspy displays two panels: one for all interactions seen over a timespan, and one for all objects subscribed. Selecting one or multiple items shows exactly what each object or interaction contains in a way that is readable to the average person.

6.1 Automated Testing

Large Federations contain hundreds of objects and interactions at any moment in time. It is virtually impossible for anyone to track all of the objects for anything but the smallest federation. The only way to test everything in an exercise is automated testing.

For the purposes of tracking all data in real time, the FOMspy provides its own rule based testing environment. Every time any information is received by the FOMspy, it goes through a list of rules to verify that the set is followed correctly.



Panel F: FOMspy Rule Editor

Rules can be defined as test agreements specified during the FEDEP process and documented in a Federation Agreement.

6.2 Rule Making

The FOMspy provides three individual methods to create rules. The first method is the easiest. The GUI provides a point and click interface that lets the user build the necessary components of a rule one by one while being virtually guaranteed that the rule is grammatically correct. Creating a rule this way follows a similar procedure as the way filters are made under most common e-mail software.

The end result of using the GUI above is a script that the FOMspy can read to understand a rule. The second method involves writing these scripts directly using any text editor. This lets programers and people comfortable writing code write the rules faster. The disadvantage is that there is greater chance of introducing errors by using this technique, but for people used to scripting this is an effective way to write rules.

The third way of writing rules is using the included plug-In system and writing a custom own DLL. This is, of course, is much more complicated than the scripts but it allows for more versatility. The only limit on rule creation is imposed by the processing speed of the computer.

Custom plug-in rule sets can be as complex as anyone can make them and can even be used to connect to databases and other applications for further testing.

As an example of its power, the FOMspy will ship with a sample rule set that logs all data of a chosen entity or entity type onto a file in the same human readable format that the FOMspy displays on the screen.

The FOMspy is intended primarily as a passive monitoring device, collecting, logging, and analyzing data, as well as providing alerts upon error conditions. In order to manage all of the information in an exercise, the FOMspy subscribes

to all objects and interactions in a passive way. This means that it should add a minuscule amount of overhead to your exercise while it is running. In turn the tool should be useful for both testing and for monitoring operational federations during actual live exercises.

7. Current Status of the Tools.

The RTIspy has been available for use with the MÄK RTI for development and debugging of federations for some time. It has been significantly enhanced by the SBIR effort.

Users have applied the RTIspy to track down data loss (e.g., is a recipient actually subscribed to a class?), to optimize code by identifying calls that dominate resource use, to make trades between explicit data declarations vs. DDM, to determine whether a problem is caused by network data losses, etc.

The enhancements described in this paper, including the WEBspy, will be made available in the near future..

The FOMspy is currently available in pre-release form. It is currently being tested in the JFCOM Joint National Training Capability (JNTC) Advanced Training Technology Laboratory (JATTL) Lab, the USAF Distributed Mission Operations Center (DMOC), and by the MITRE Corporation within large federation exercises available to them.

References

[1] United States Joint Forces Command: "Millenium Challenge 02", http://www.jfcom.mil/about/experiments/mc02.htm

[2] Ceranowics, A, et. al.: “Reflections on Building the Joint Experimental Federation”, Proceedings, I/ITSEC 2002.

[3] DARPA Defense Sciences Office: "Training Superiority (DARWARS)," http://www.darpa.mil/dso/thrust/biosci/training_super.htm

[4] NASMP Systems Engineering Federation Working Group: "Navy Aviation Simulation Master Plan Federation Agreements Document (FAD)," NAVAIR Orlando, 2004

[5] Defense Modeling and Simulation Office: "HLA Federate Compliance Testing," https://www.dmso.mil/public/transition/hla/compliancetesting

[6] Defense Modeling and Simulation Office Software Distribution Center: https://sdc.dmso.mil/

[7] SITH Web Site, http://www.openchannelfoundation.org/projects/SITH

[8] FMT Web site, http://fmt.mak.com

Author Biographies

Felix Rodriguez is a software engineer at MÄK Technologies. He serves as Principal Investigator for MÄK’s work on the RTIspy and FOMspy. He is also responsible for the FMT transition to the new 1516 RTI architecture.

Erik S. Hougland, PhD is a Computer Engineer with the Naval Air Systems Command Training Systems Division Orlando, Florida. His responsibilities include fielding test tools for Federation Interoperability testing and Synthetic Natural Environment use.

Keith Nielsen is a Computer Engineer with Naval Air Systems Command Training Systems Division Orlando, Florida. He is an interoperability engineer on the Navy Aviation Simulation Master Plan.

Miles R. Fidelman is Business Development Manager for MÄK Technologies’ technology programs. He has 30 years experience in systems architecture and engineering for data networks and distributed computer systems.

Brian Spaulding is Director of Contract Engineering at MÄK Technologies. In this role, he provides technical direction for MÄK’s technology programs.