REFLECTIONS ON BUILDING THE JOINT EXPERIMENTAL FEDERATION

Andy Ceranowicz
IITRI

Mark Torpey, Bill Helfinstine, and John Evans
Lockheed Martin Information Systems

Jack Hines
Titan Systems Corporation

ABSTRACT

The Joint Experimental Federation (JEF) is a large collection of simulations federated together by the US Joint Forces Command (JFCOM) and the Military Services to support Millennium Challenge 2002. The federation stimulates Service C4I devices to create a Common Operating Picture for a Joint Force Headquarters. The MC02 construct also included concurrent Service experiments, such as the Navy's Fleet Battle Experiment and the Air Force's Joint Expeditionary Force Experiment.

JFCOM asked the Services to nominate simulations that would drive their C4I systems and realistically portray military capabilities. The JEF was constructed by linking the nominees together via the Defense Modeling and Simulation Office's Run Time Infrastructure (RTI). The resulting federation linked RTI and Distributed Interactive Simulation (DIS) federates and live exercise feeds into a single geographically distributed simulation. Federation integration took a year and was primarily a process of resolving conflicting simulation approaches. Significant extensions allowed RTI-1.3NG to achieve the scalability and fault tolerance required for MC02. The federation simulated over 34,000 battlefield platforms over a Wide Area Network and allowed individual simulation federates or communications links to fail and restart without restarting the entire federation. A common Federation Object Model (FOM) and Federation Agreements were developed based on the Real-time Platform Reference FOM.

While federations allow for simulation reuse, they also have drawbacks. Interactions gravitate to the least common denominator models and the effort required to implement new capabilities is significantly multiplied. The real power behind the federation concept is that it allows disparate groups to use their own simulations in common experiments.

ABOUT THE AUTHORS

ANDY CERANOWICZ was the Technical Lead for the MCO2 JEF. He led the development of the FOM and Federation Agreements. Andy is a Senior Science Advisor at IITRI in Harvard MA.

MARK TORPEY was a MC02 federation developer and is the lead developer and integrator of JSAF (Joint Semi-Automated Forces). He is a Staff Software Engineer at Lockheed Martin Information Systems Advanced Simulation Center (LMIS-ASC) in Burlington MA.

BILL HELFINSTINE was a MC02 federation developer. He is the lead developer for JSAF RTI interfaces and RTI-S. He is a Staff Software Engineer at LMIS-ASC in Burlington MA.

JOHN EVANS was a MC02 federation developer. He developed the enumeration testing tools and monitored enumeration compliance. John is a Senior Software Engineer at LMIS-ASC in Norfolk VA.

JACK HINES was a federation developer for MC02. He developed the detonation testing support tools. Jack is a Senior Systems Analyst at Titan Systems Corporation in San Diego CA.

REFLECTIONS ON BUILDING THE JOINT EXPERIMENTAL FEDERATION

Andy Ceranowicz
IITRI

Mark Torpey, Bill Helfinstine, and John Evans
Lockheed Martin Information Systems

Jack Hines
Titan Systems Corporation

THE FEDERATION

Millennium Challenge 2002 (MC02) was a large simulation supported experiment conducted in July and August of 2002 by 13,500 personnel at locations across the United States. It was led by US Joint Forces Command (JFCOM) under a mandate from Congress to demonstrate the Joint and Service warfighting concepts described in Joint Vision 2020. In addition to JFCOM and the Military Services, MC02 also included participation by a number of other government agencies. JFCOM focused on exploring new concepts for a standing Joint Force Headquarters executing Rapid Decisive Operations. The Joint Experimental Federation (JEF) was designed to simulate the Joint battlespace for the purpose of presenting a realistic operational picture to stimulate the experimental audience. The JEF also supported Service specific experimentation such as the Navy Joint Fires Initiative and the Air Force Global Strike Task Force.

Table 1. provides a list of the simulations used in the JEF (USJFCOM 2002). This list only includes simulations electronically connected to the federation. A number of additional simulations were run offline to simulate other required functions, such as the flow of forces into theater and the handling of casualties.

Table 1. Simulations in JEF

SERVICE	SIMULATION	INTER-FACE
Air Force	AWSIM	RTI
	CATT	RTI
	DICE	RTI
	GIAC	RTI
	DCE	DIS
	IWEG	DIS
	JQUAD	DIS
	PSM+/NAV	DIS
	MDST	RTI
	VSTARS	DIS
	AFSERS	DIS
	UMS	DIS
	CL-GPS	DIS
	AWACS	DIS
	COBRA BALL	DIS
	Airborne Laser	DIS
	Rivet Joint Trainer	DIS
	CV-22	DIS
	LOGSIM	Indirect
ARMY	JCATS	RTI
	EADSIM	DIS
	FIRESIM	DIS
	TUAV	DIS
	TENCAP MUSE	DIS
Marines	JCATS	RTI
Navy	JSAF	RTI
	WARCON	RTI
	OASES	RTI
	TENCAP MUSE	DIS
	WALTS	RTI
	IMPACT (CUSP)	RTI
	PC SWAT	RTI
	BFTT	DIS
	AUTO SIGS	DIS
	ATLOS	RTI
	LEAPS	DIS
	vSSGN	DIS
NRO	SLAMEM	RTI
	NWARS	RTI
SOF	JCATS	RTI
	JSAF	RTI
	AWSIM	RTI
Joint	Civil Env. Model	Indirect
	JSAF Clutter	RTI

The High Level Architecture (HLA) is the DoD approach to simulation interoperability (Kuhl 1999). Simulations using the HLA communicate via a Run Time Infrastructure (RTI). RTI simulations were connected directly into the federation. Distributed Interactive Simulations (DIS) use an older interoperability standard (IEEE 1998) and were connected to the federation via gateways. The Civil Environment Model and LOGSIM were indirectly connected via other simulations. There were too many simulations in the MC02 JEF to allow more than a brief mention of the primary ones. JSAF was used to model naval and maritime platforms, including Navy air and shore based anti-ship missile batteries. JSAF also provided civilian traffic referred to as clutter. AWSIM represented Red and Blue air, as well as civil air traffic. CATT, DICE, IWEG, and DCE represented the Opposing Force (OPFOR) integrated air defense. JCATS modeled the land forces for the Army, Marines, SOF, and OPFOR. JQUAD provided reporting for AWSIM entities and modeled fixed targets. SLAMEM and NWARS modeled national sensors and MDST modeled the detection of OPFOR missile launches.

RECONCILING DIFFERENCES

The process of building a federation from existing federates is primarily one of reconciling differences in the philosophy and implementation of the simulations involved. Our integration began in June of 2001 and continued until experiment execution in July of 2002. The majority of our integration efforts could be classified as:

- Porting federates to HLA
- Maturing the RTI and network
- Reconciling differences among federates

These tasks consumed virtually all our available time leaving little to implement new federation functionality. Because the majority of the MC02 simulations had previously supported DIS, we had a common base to start from. Even so, there were unexpected conflicts in the interpretation of fundamental DIS interactions. For example, when should entities be deleted from the simulation? AWSIM simulates aircraft only while they are flying. Once they are destroyed or land 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.

ENTITIES VS. AGGREGATES

The first question resolved was whether to use aggregate or entity representations. Since we were dealing with the operational level of warfare, there was a strong argument for aggregate representations. However, since a primary focus of the experiment was C4ISR assets, which detect individual entities not aggregates, entities made more sense. The scenario also focused on small units rather than large formations, further limiting the usefulness of aggregate representations. The major question was whether we could simulate and control enough entities to support the scenario. We targeted 30,000 entities for our federation and actually used approximately 35,000 in the experiment. Although that was not enough to model all the civilian traffic in the theater, it was sufficient to represent all of the entities tracked via the C4I devices used in the experiment.

TIME: MANAGED VS. REAL

The decision to run a real time federation without time management was made early. In fact, we made simulation time identical to wall clock time to support interfaces with real world C4I equipment and live forces. We would have preferred to run 12 hours of simulation time per wall clock day, but getting the C4I equipment to adapt to discontinuous simulation time was decided to be too hard. So we used a 24/7 simulation execution schedule.

There were other reasons for using a real time federation without time management (Hung 2002). An important characteristic of real time simulation is that simulation time advances without requiring overhead for coordination. In time managed federations, a time advance request and grant procedure is required to move time forward. Time management overhead increases with the number of federates and smaller time steps. The JEF included around 90 federates with internal update rates that ranged from 0.1 to 10 seconds, so the overhead for advancing time could have been significant. The JEF computers do, however, use NTP to synch their system clocks. Real time simulation supports fault tolerance as well. Since time management essentially waits for each federate to catch up, if a federate gets into a state where it cannot proceed forward or cannot communicate its advance, the entire simulation stops. This was unacceptable for an experiment that would be run once and included live forces. Finally, since the majority of the simulations in the federation had DIS roots, they were already designed for real time. Not using time management just made it easier to interface the DIS models to the federation.

Arguments for using time management include reproducible simulation runs, running faster than real time, and insuring proper event ordering. In this case, however, we were executing only a single trial and the C4I equipment and live operators restricted us to real time. Event ordering anomalies occur when external events are processed at a time later than when they occur due to publishing, transmission, or federate processing delays. Thus a tank in one federate may kill a remote tank before it can open fire, but because of delays in processing the detonation, the second tank gets a chance to fire back. In this federation these delays were usually under two seconds, so for these anomalies to occur, both tanks need to fire within a couple seconds of each other. Given the C4I focus of the experiment these types of effects were of negligible consequence.

FOM: BOTTOM UP VS. TOP DOWN

Our first major activity was to determine what should be in our Federation Object Model (FOM). The FOM is an HLA formalism for capturing all the object classes, attributes, interactions, and parameters, which will be used to communicate simulation state between federates. There were two potential approaches. You could work bottom up from a blank FOM and add in only the data required for the current federation. Alternatively, you could start with an existing FOM from each simulation and work top down to resolve the conflicts. We used the second approach. We started with the core set of federates: AWSIM, DICE, CATT, MDST, JQUAD, JCATS, and JSAF and merged their FOMs. We guided conflict resolution by defining what each federate would simulate and what interactions would be supported. There was a great deal of overlap in simulation capabilities. We divided the FOM into public and private components. Public components were used by all federates, while private components were only used between federates from one or two simulations. For example, OASES used environmental objects to communicate ocean state to JSAF. This top down approach was probably the most efficient because of several factors. First, since our FOMs were based on the Real-time Platform Reference FOM (RPR) (Reilly 1999), there were few cases that needed to be resolved. Most often, federates had only to add FOM items that they had not supported in the past. Second, because these were existing simulations, they already supported other distributed simulation applications. Allowing them to keep most of their existing FOMs intact minimized conflicts with their original applications. Private protocols could be merged in without discussion since, by definition, private protocols had no conflicts. The disadvantages of this approach were that the JEF .omt file, which defines the FOM, was over a megabyte in size and provided no explicit distinction between private and public data elements.

FEDERATION AGREEMENTS

Significantly more information is required to create a working federation than what is in the FOM. The HLA defines the FEDEP process (DMSO 1999) to capture this information. However, we decided to follow the example of the RPR GRIM document (Reilly 1999) and developed a set of federation agreements. But, we disagreed on how to document them. Some favored a document describing differences from the RPR, but we ended up writing an independent federation agreements document. While this worked reasonably well, in hindsight, we feel it would have benefited from more detail than was provided and we continue to disagree.

DESIGN: SERIAL VS. PARALLEL

Questions about operational requirements came up continually during the design of the FOM. What would the scenario look like? What would the force list be? The reality was that the time available to create the federation precluded waiting for other groups to formulate their requirements. The experimental concepts were still in flux and the force lists and scenario delivery dates were far away. The only approach that would produce an experiment in the available time was to work in parallel, make assumptions, and adjust as requirements became more solid. While this is not the most efficient approach in resource utilization, it is the fastest way to develop and mature concepts.

RTI

The major part of our federation development time was spent porting simulations from DIS to the RTI and maturing the RTI to support the experiment. The RTI is the common network interface code that federates use to communicate with each other. We started the process with two candidate RTIs. The primary candidate was the DMSO 1.3NG RTI (RTI-NG) (Bachinsky 1999) and the secondary was RTI-S (Calvin 1997), developed by DMSO and DARPA for the STOW Program. The DMSO team was able to modify RTI-NG sufficiently to support MC02 requirements and it was used for the experiment. The availability of two different RTIs was extremely useful during development, as switching between them helped to diagnose problems and suggested alternative solutions.

Subset vs. Complete RTIs

RTI-NG is a fully compliant RTI that implements the entire RTI-1.3 Specification (Spec) (DoD 1998). RTI-S

predated the Spec and had been engineered to provide those features required to support large-scale real-time federations. RTI-S supports only a portion of the Spec but provides additional services not required by the Spec. Functions such as time management are not fully implemented, while minimum rate transport modes are provided. RTI-NG was developed with a strict interpretation of the Spec. Because many parts of the Spec were stated in the form: "if service X is invoked on federate A, then action Y SHALL be executed on federate B," RTI-NG implemented all these functions using reliable protocols and centralized control. Since this led to severe performance and fault tolerance problems for the federation, we proposed that DMSO allow federations to use subset RTIs, like RTI-S. DMSO rejected this approach on the basis that allowing subset RTIs would limit interoperability and be too hard to manage effectively. To be compliant, an RTI has to implement all the functions in the Spec. Fortunately, RTI-NG can modify its characteristics via the Run-time Initialization Data (RID) file. A user can specify RID parameters to configure the RTI for his federation. Eventually DMSO allowed the RTI-NG developers to add RID parameters that would enable RTI-NG to support a large real-time federation effectively.

Porting to HLA

Most MC02 simulations started federation development with DIS interfaces. To move forward while federates were building their RTI interfaces, we used gateways to link their DIS versions into the federation. As each simulation was ported to HLA, it was moved from the DIS side to the HLA side. This approach was very important to our progress because it allowed simulations to participate in the federation immediately. They could deal with other integration issues such as enumerations and it allowed the team to gain confidence that the federation could work early on in the development process. Basing our federation on the RPR FOM made porting from DIS and developing gateways easier.

Outboard vs. Inboard RTIs

Two approaches were used to connect to the federation. JSAF, SLAMEM, MDST, and DICE used inboard RTIs. They used the RTI for their private communications as well as federation communications. When running multiple JSAF simulations, each one joins the federation as a separate federate and the control messages from one JSAF to another go through the RTI. On the other hand, many of the server style simulations like JCATS, CATT, and AWSIM used multiple computers communicating with internal protocols and had a single computer acting as a gateway or bridge to the RTI federation. All the computers for one of those simulations were treated as a single federate. Thus the number of computers in the federation significantly exceeded the number of federates. Centralized architectures with a gateway typically could not take advantage of Data Distribution Management (DDM). This is an approach for limiting the network traffic each federate has to process to provide scalability. DDM works by segmenting federation network traffic so that federates can listen to only those segments containing data about objects and interactions that can affect them. A gateway, which connects many kinds of entities and sensors scattered throughout the playbox to the federation, makes it impossible to segment the simulation traffic because the simulations behind the gateway are interested in all of it. Using a gateway can also increase overall network bandwidth requirements since the same information is transmitted in multiple formats. On the positive side, the use of a gateway or bridge does allow a simulation to use more efficient specialized protocols for its internal communications and provides isolation from FOM changes.

FAULT TOLERANCE VS. RELIABILITY

One of the biggest RTI issues that we had to face was fault tolerance. As a federation tasked to support a one-time exercise with many live players, we had to do everything we could to keep the simulation running and to recover seamlessly even if individual federates failed. As long as a federate can recover before the C4I picture is compromised, the federation can keep running. Thus we had to prevent the failure of individual federates or network connections from causing a failure of the federation.

On first glance, fault tolerance and reliability are compatible and desirable features, but on deeper inspection they turned out to be polar opposites. Fault tolerance means that if an individual component fails the remainder of the federation keeps on running. Reliability of message delivery means that the RTI must guarantee that the message is received. This is typically done by buffering messages and resending them until an acknowledgement is received. If no acknowledgement is received, federates can run out of memory and crash in turn, bringing down the entire federation. Initially, even stopping a single federate in the debugger would crash the federation. The RTI-1.3 Specification has no requirement for federation fault tolerance but its language does imply reliable message delivery. So the failure of any federate can cause the failure of the federation. This is not unreasonable if you assume losing a federate invalidates the experiment, however for our application it was the wrong

assumption. To work around this, a RID parameter was provided that turned RTI-NG into a connectionless RTI without a central RTI executive, supporting only best effort transmission. The magnitude of the change inspired the humorous name "hurt_me_plenty" for the parameter. With this change, the majority of the fault tolerance problems were resolved. As we migrated from a LAN to a WAN environment this proved to be critical. Federates started to crash when they received packets corrupted by the network and we were unable to find and correct the source of the problem. We eventually worked around it by adding CRC checking of packet payloads in the RTI. Corrupted packets were discarded allowing the federates to survive.

HEARTBEATS VS. QUERIES

The remaining federation stability problem was due to network spikes. In most cases, we traced these problems back to class queries. Research on scalability had suggested that you could limit traffic by publishing static information once and having federates that missed the information query for it. The Spec requires support for both class and individual object queries. Unfortunately, with over 30,000 entities in the federation, a query for the base class results in a very large traffic spike. Since queries have a nasty tendency to become unstable, we disallowed queries in the Federation Agreements. We used a special RTI module to detect queries and identify offenders. We also turned off as many individual attribute queries as possible. While these did not cause large spikes, requesting attributes for 30,000 entities individually, requires a lot of queries. In place of queries, we went back to the old DIS practice of heartbeating objects periodically even if their attributes have not changed. This traded large spikes for an increased average traffic rate. We found that the smoother the network traffic rate was, the more stable the federation was. Heartbeating also allowed federates to time out objects when they went out of scope. The RTI-NG advisory mechanism for tracking scope was not scalable.

SCALABILITY: RTI VS. SIM

The other major RTI issue was scalability, that is, how many entities could the federation simulate if more computers could be added. We had a operational requirement to simulate at least 30,000 individual entities. Initially, with RTI-NG, the federation was only able to handle several thousand entities.

Packet Rates vs. Entities

We characterize federation performance by entity count because scenario developers are interested in the size of the forces they can represent. However, RTI performance is more tied to attribute update and interaction rates and sizes than to the specific number of entities. Also the actual number of RTI objects required to support these entities was much larger that the entity count. Throughout the integration process we worked on reducing the message traffic required to support 30,000 entities. Our initial measurements indicated a potential traffic rate as high as 48 megabits/sec and we were able to reduce it to less than ten. Another important dimension to scalability is the number of federates the federation can support. Our initial RTI configuration maintained a mesh of reliable connections between federates. The overhead of maintaining this net increased with the number of federates. Going to a connectionless RTI improved federate count scalability by eliminating the reliable mesh and globally shared RTI knowledge. This lowered packet rates by reducing RTI administrative traffic.

We also reduced federate update rates. A federate publishes updates if attribute values change or if heartbeats are required. For continuous attribute values such as position and velocity we used the concept of dead reckoning: the attribute is updated only if the error from its dead reckoned value exceeds a threshold. Initially we ran the federation with DIS threshold values and heartbeat rates and then we gradually relaxed them. We ended up with a 60 second heartbeat and thresholds of 10 meters in position and 15 degrees in rotation. We also eliminated attributes, such as turret orientation, that would significantly increase traffic and were not required for the experiment.

The packet handling efficiency of RTI-NG was optimized and we tuned the packet bundling parameters to get better performance. We bundled up to 4500 bytes in each IP packet and collected updates for up to one second. The tradeoffs were that bundling more data together increased latency and packet loss due to transmission errors, while smaller packets increased the transmission of overhead data.

Declaration vs. Data Distribution Management

The other approach we took to scaling the federation was to filter traffic going to each federate. The RTI provides two mechanisms for filtering: Declaration Management (DM) and Data Distribution Management (DDM) (Helfinstine 2001). DM is a more static approach; once you publish an object to a class you cannot change it. Each federate can then tell the RTI which classes it wants to receive at any point in time. DDM allows information to be filtered more selectively and dynamically. DDM allows you to publish any object or interaction to a space of publication

dimensions. Federates can select a region of the space to publish to and subscribers can select a region of the space to listen to. If a publication space overlaps a subscription space, the subscriber will receive the update or interaction. Unlike DM, DDM allows a federate to change the region it publishes to, in order to indicate that the object has changed its properties and that it may be of interest to a different set of listeners.

Different federates approached filtering from different perspectives. JCATS used very little filtering since it had a server architecture with a gateway bridge. Since its entities were scattered over the playbox, there was little benefit to filtering. Other federates favored DM. AWSIM filtered out ground entities except for a special class that included missile launchers and anti-aircraft systems. That allowed it to ignore the bulk of the updates, which came from the other ground vehicle classes. The drawback to using DM for filtering is that it is static. An entity cannot become interesting because it has moved into your sensor range and you cannot have radios publish to different classes based on the current frequency they are using unless you delete and recreate the object. If you subscribe to ground vehicles, you get all ground vehicles from the entire playbox because, in practice, you cannot subclass on location. If you did, the entities would not be able to leave the region specified in their original subclasses. DDM allows you to filter on dynamically changing attributes such as position, frequency, and emissions.

Software vs. Hardware Filtering

Filtering for DM and DDM was implemented in RTI-NG in three ways. First, if no other federate subscribes to information published by a federate, the RTI can apply source based filtering and never send it out on the network. However, in practice, with 90 federates in the federation, virtually everything will have at least one federate subscribing to it, if only to log it. This type of filtering was not used, and the advisories that supported it were eliminated when we went to a connectionless RTI.

The second type of filtering was based on the comparison of publication and subscription regions. Each update had its publication region sent with it. Upon arrival at a federate, the publication region was compared with the subscription regions of the receiving federate and the update was reflected to the application only if there was an overlap in the regions. While this worked acceptably for a federate which used very few regions, the comparison was enormously expensive for a federate that used many regions. Additionally, including the publication region with each update increased the size of the publication and therefore the network bandwidth required to support the federation. Another RID file parameter was provided to turn off this region comparison.

The final form of filtering, and the only one we actually used was multicast filtering. This filtering is primarily accomplished in the network hardware, either in the network switches and routers or at the network interface card. Thus, it can significantly reduce the filtering performed by the federate processor. While all RTI federates used DDM to take advantage of multicast filtering, not all federates used it dynamically. When using DDM, simulations request that the RTI publish data to regions in a space specified by the federation (Coffin 1999) For MC02, the space consisted of a primary dimension that was divided into 31 enumerated regions, such as the Blue air space, the Red ground space, the radio space, and the detonation space. Two additional dimensions were used to further separate objects published to each region on the primary dimension based on their attributes. For example, the Red ground space interprets the two secondary dimensions as geographic extents, so the position of the entities determines their publication region. Publications to the radio space treat one secondary dimension as a frequency dimension and ignore the second dimension. Publications to the detonation space ignore both secondary dimensions. Thus a receiving federate can subscribe to Red ground entities in a particular area, or it can subscribe to radios in a particular frequency range, but if it subscribes to detonations it will get all of them.

To turn these regions into multicast addresses, the RTI is given a DDM scheme that tells it how many multicast addresses to assign to each dimension. For the Red ground space, 15 multicast cells were assigned to each secondary dimension producing a grid of 225 multicast addresses over the playbox. The detonation space is assigned a single address. Using this information the RTI takes the publication region specified by the publisher and assigns it to a single multicast address for publication. Publication regions were restricted to point regions to avoid overlapping multiple multicast address regions. Subscription regions, which are typically not points, translate into subscriptions to the multicast addresses covering the region. This allows the receiving federate to tune into the data it needs just like tuning to a number of TV channels. MC02 used slightly more than a thousand multicast addresses for all its publications.

A number of new RTI features had to be implemented to make this filtering scheme practical. First, we had to stop federates from publishing without DDM extents. RTI-NG will take a non-DDM publication and publish

it to every multicast address. This ensures that all DDM subscribers receive DM publications. For federations that use a few multicast addresses, this is not harmful. But for our 1000 multicast addresses, this multiplied the bandwidth required for DM publications by 1000 and caused spectacular network spikes. Forcing federates to publish using DDM with point regions eliminated this multiplication. Initially, RTI-NG did not allow efficient allocation of multicast addresses. It would only allow a uniform grid over all the dimensions of the publication space. Thus if you used 225 addresses for the Red ground space, you had to use 225 address for every other region on the primary dimension. With 31 regions on the primary dimension, we had to use 6,975 multicast addresses to get a 15x15 grid over the playbox. Nonuniform gridding was added and brought this number down to around 1000. Multicast processing was also initially very inefficient in RTI-NG. Initializing large numbers of multicast addresses took many minutes and sometimes did not work at all. Through repeated code optimization, these problems were overcome and it became practical to use DDM.

ENUMERATIONS

In order for federates to understand each other they need not only a common FOM, which is akin to a grammar, but also a common vocabulary. That vocabulary is provided by enumerations. Enumerations are used to identify platforms such as an M1A2 tank, munitions such as a Block 2 Standard Missile, radar emitters, and IFF codes. While conceptually simple, developing a common set of enumerations and making sure that they are implemented in each simulation requires considerable patience and constant attention. MC02 probably posed a greater challenge than most federations because of the sheer number of federate types and the constantly changing enumerations. We started with around 1000 platform enumerations based on the DIS standard (Braun 2000). Within weeks of experiment execution we were still tweaking enumerations, primarily to accommodate changing force lists. To help with this task, we built a number of tools to partially automate testing. The first of these tools was an enumeration blaster. It was a federate that could publish an array of all the entities with legal enumerations. This tool was used to check whether federates would recognize the enumerations correctly. Unfortunately verifying this was a painful manual process. While it was useful for initially checking a federate, the dynamically changing entity list prohibited its continued use to verify changes. Fully automated checking would have been useful, but that would have required a second independent way of identifying the entity and instrumentation to respond to automated queries. Most federates did not have the time or resources to implement additional instrumentation. We also provided a tool, called SNN, to monitor the federation and detect enumerations that were not on the legal enumerations list. This was used for individual tests and during integration events. SNN monitored all enumerations published and produced a list of all illegal enumerations and the simulations that produced them. Unfortunately, SNN could not determine whether the enumerations that were on the legal enumeration list actually represented the correct platform types. With these tools and much manual effort, the enumerations were slowly synchronized over the integration period. One of the things that made the problem easier was that the enumeration blaster, SNN, gateways, JSAF, and clutter all shared the same enumeration file. If that sharing could be extended to more simulations it would be very helpful.

DAMAGE ASSESSMENT

The most difficult set of enumerations to verify were munitions. Verifying them required examining their damage against each platform they would be used to attack. This is a cross product problem where the brute force approach would try each weapon against each platform in the federation. Not only were there many cases, but, because damage is calculated probabilistically and depends on many factors, such as the velocity and incidence angle of the hit, the number of combinations gets out of hand rapidly. Recognizing that this was a significant federation problem, we attempted to add supporting tools and instrumentation. We directed each of the federates to implement a damage assessment interaction that would report data such as the probabilities of different levels of damage and any enumeration mapping which the target simulation was using for each detonation it received. Then we developed a detonation blaster that would hit each entity in the federation with a series of detonations and modified SNN to capture the resulting damage assessments. The SNN log was processed to find problems such as detonations that were thrown away because the federate did not know about the munitions or did not have damage tables for those munition/target pairings. We also requested the implementation of a mode in which detonations produced damage assessments but did not actually implement the damage. This allowed the detonation blaster to try a series of detonations against each entity without having to replace damaged vehicles. For each run with these tools, a federate would create an array of all the entities it was going to simulate in the experiment and the detonation blaster would try each of the legal munitions against it with a variety of detonation result codes and detonation miss distances. Detonation results are

enumerations that indicate whether the detonation represents a direct fire hit, a proximate hit, a miss, or an indirect fire detonation. The result of the test was a list of detonations together with the resulting damage assessments describing how the detonation was processed including rejection cases, munition substitutions, and raw damage probabilities. Special cases that did not yield expected outcomes were flagged for easier recognition. Unfortunately, the analysis program did not have a table of expected damage probabilities. So the problems it flagged were limited to the processing preceding the actual damage calculation, such as, munition mappings and the handling of damage result codes. Even those problems required someone to determine which ones needed to be corrected and how. Although a few simulations implemented the damage assessment, most did not have the time or resources to do so. However, we were able to use the precise control of detonation parameters provided by the detonation blaster to run through a set of general test cases for manual analysis. For those that implemented damage assessment, we found that the blaster generated huge numbers of cases for Subject Matter Expert (SME) review. Although the blaster had some simple logic for eliminating target munition pairs that were unlikely to occur, it was still quite exhaustive in generating test cases. Lacking instrumentation for most federates, we had to rely on a manual review of the damage tables. We limited the scope of the manual review by considering only cross simulation interactions and selecting only the likely target/munition pairings. The review focused on identifying pairings that were ignored or had munition substitutions. The SMEs for the firing simulation were required to determine whether the substitutions were acceptable or that pairing could be ignored. This approach was coupled with face validation testing for major interactions. Unfortunately, we could only scratch the surface of this problem. There was just no practical way to verify that damage was correct under all conditions.

One of the surprising results from the general test cases generated by the blaster was that even though most of the simulations used the same DIS paradigm for damage and had employed it in previous DIS exercises, their interpretations of the model were significantly different. Some simulations responded only to direct fire. Some required that the firer parameter field be filled in. Others treated proximate misses as complete misses, while still others treated them as direct hits. These kinds of problems could not have been found by damage table analysis alone. The detonation testing experience showed the true difficulty of performing a thorough V&V of such a large federation. It cannot be accomplished without automated testing, which in turn requires the allocation of sufficient resources for instrumentation and SME support.

FEDERATION MONITORING

A very important instrumentation effort allowed us to monitor federation health. We needed a way to determine whether the entities each federate was simulating were visible at all the federation sites. We also needed to be able to identify which federate was simulating a particular entity. To accomplish this, we came up with a format for the RTI object name that identified the federate simulating the entity. Each RTI name contained a string identifying the simulation, the federate IP address, and a locally unique ID for the entity. DIS entities could not be identified from their RTI object names because they contained the name and IP of their gateways. To allow the source of these entities to be identified, we assigned site numbers to each simulation type. These DIS IDs were preserved in the HLA object identifier attribute, so gateway objects can have their DIS IDs examined to determine their sources. To make use of this instrumentation, we developed an SNN operating mode in which SNN subscribes to all entities and displays the number of entities from each federate, HLA and DIS. By placing SNNs at each site, we could see if all the sites were receiving the same number of entities. This provided us with a good indication of network and federation health. We also had RTI commands to tell us which federates were joined to the federation without running a central executive process. We did not have sufficient time to implement more instrumentation, but what we developed was crucial to our ability to monitor and control the federation.

TERRAIN CORRELATION

The MC02 federates used a wide variety of terrain representations. AWSIM and JQUAD used a flat Earth geodetic projection. DICE and CATT used geodetic coordinates with stair step elevations. JCATS had a gridded Cartesian coordinate system augmented with elevations. JSAF had a coordinate system that tiled the Earth with Cartesian cells and used an irregular grid of triangles to represent the elevation surface. Getting all these representations to agree was a major challenge. We started by attempting to use ground clamping to provide correlation. As each remote entity position was updated, the local copy of its position was adjusted to place the entity on the surface of the local terrain or ocean. Of course, this was not possible for air and subsurface entities. We considered adding a surface relative altitude attribute so that we could adjust the local elevation to match the altitude in the owning simulation. This approach had a number of

disadvantages. It was computationally expensive, especially if the remote entities were distributed enough to require terrain to swap in and out. Clamping could produce very choppy aircraft movement since the aircraft would essentially bob up and down with the local terrain surface. Fair fight issues would emerge because entities hidden by terrain features in one federate would be exposed in another. Finally, dead reckoning would not work since it was driven by different terrain surfaces. Thus we decided it was necessary to try to correlate all the terrains. We started by building a single source data set. Then we generated all the other terrain representation formats from that source data. Both JCATS and AWSIM underwent major terrain representation changes, with AWSIM adding elevation data to their terrain model and JCATS changing from a projection to round Earth representation in a Cartesian space. JSAF built a terrain representation with regular triangles to exactly match the JCATS grid. DICE added geoid corrections to its altitudes. The resulting terrain correlation was fairly good with most elevations being within a couple of meters of each other. However, there was no way to perfectly align all the disparate coordinate systems. Fortunately, almost all the ground forces were in the JCATS federates, which by definition had perfect correlation. No terrain anomalies were noticed during execution.

LEAST COMMON DENOMINATOR

One of the difficulties of creating federations is that interactions between federates must be grounded in common or at least compatible models in all federates. So the tendency is not to use the most sophisticated models, but instead to pick a simple model that all federates can understand. For example, when modeling day and night, instead of using a gradual change in illumination we used an on/off switch. Similarly when dealing with terrain features, some federates had abstract vegetation while other federates had explicit representations of individual trees. We ended up ignoring vegetation because it took so much effort just to get the elevations correlated. In the case of camouflage, where we had to get everyone to implement a new model, we used a very simple four state model with percentage degradations in the probability of detection for each camouflage type. Thus, federations that integrate platforms from different federates tend not to make use of the more sophisticated features of any federate but instead gravitate to the least common denominator models in the federation. Because of the many DIS models we were using, we were even further constrained. The approach we took, integration by platform, can be described as vertical integration. An alternative approach would be to integrate horizontally. Instead of building each platform or entity entirely in one simulation, one federate could provide the detection model, another the targeting model and so on. This would get around the least common denominator problem and fair play issues. However, this approach brings out an entirely new set of problems such as increased communications among federates, more latency issues, much more complicated interfaces between models, and a complete inability to test without every federate present. So, it is unlikely that this approach would be easier.

MULTIPLICATION OF WORK

Vertical integration requires that a model of each capability needs to be implemented in every simulation. For example, every major simulation has its own model of detection. If you decide to improve a common model or add a new one, you have to do it in every simulation. This multiplies the effort required to upgrade and evolve the simulation. While there are a few models that are Service specific, like ship hull motion, most, like radar, are common to all the simulations. It is necessary to not only build a new model once for each simulation, but also to negotiate a design that every simulation can support. This makes evolving a federation like the JEF expensive. Again, horizontal integration might solve this problem but at the cost of many new problems. New technology would probably be required. A server architecture where only one federate executes a particular model may not be appropriate for horizontal integration. Perhaps model code could be dynamically distributed to federates that need it. While horizontal integration is not practical today, it is probably worth exploring approaches to horizontal integration for the long term.

HUMAN NATURE

The rationale for simulation federations has been that they allow reuse of simulation code and therefore make simulation much more economical. However, our experience indicates that for experimentation, this idea can break down. Experimentation requires rapidly changing federations to try out new concepts. The more simulations you have involved, the harder it is to modify the federation to simulate future weapons, tactics, and effects. However, federations do provide very practical advantages. For future Joint experiments, the cooperation of the Services, government agencies, and coalition partners will be required. To represent themselves accurately, these groups will often prefer to use their own models. It is difficult to learn a new simulation and understand its capabilities, limits, and workarounds. It is also very difficult to design and implement a common simulation

for many parties. The effort required to coordinate the requirements and design can easily exceed the implementation cost. Federations allow each group to design and build simulations independently and then take advantage of the best of them in a Joint context. It is easier to get everyone to pull together if they can bring their own simulations to the experiment.

CONCLUSIONS

The Millennium Challenge 2002 team has created one of the largest federations ever attempted. Over 40 simulations have been linked together via the RTI and HLA/DIS gateways. Added to these simulation federates are monitoring, recording, and C4I interface federates. In addition, live forces from all Services have been integrated into the experiment. Putting together this federation has taken a lot of work by many talented people. We did not run into any showstoppers or fundamental roadblocks in the integration. The federation became steadily better the longer we worked on it and we could, of course, continue to improve it. However, our experience has raised questions as to the efficiency of the vertical federation concept and whether technically it provides the best simulation for the effort expended. Horizontal integration may turn out to be more effective in the long run. Nevertheless, federations are a great mechanism for bringing together disparate communities into a common experiment.

ACKNOWLEDGMENTS

The authors would like to thank the JFCOM federation management team for handling the political, financial, and operational coordination required to create and operate the MC02 federation. The efforts of Tony Cerri, Annette Ratzenberger, CDR Jeff Connor, Todd and Terri Morgan, LTC Jeff Kulp, Rae Dehncke, Donna Brooks Dehncke, plus many others were critical to the success of the MC02 federation.

REFERENCES

Bachinsky, S., Noseworthy, R., & Hodum, F. (1999). Implementation of the Next Generation RTI, *Simulation Interoperability Workshop*, 99S-SIW-118, March 1999.

Braun, J. (2000). *Enumeration and Bit Encoded Values for use with Protocols for Distributed Interactive Simulation Applications,* IST-CF-00-10 August 13, 2000. See http://www.sisostds.org/doclib/doclib.cfm?SISO_CID_41.

Calvin, J., Chiang, C., McGarry, S., Rak, S., Van Hook, D., & Salisbury, M. (1997). Design, Implementation, and Performance of the STOW RTI Prototype (RTI-s), *Simulation Interoperability Workshop*, 97S-SIW-019, March 1997.

Coffin, D., Calef, M., Macanucco D., & Civinskas, W. (1999). Experimentation with DDM schemes, Simulation *Interoperability Workshop,* 99S-SIW-053.

DMSO (1999). *High Level Architecture, Federation Development and Execution Process (FEDEP) Model*, Version 1.5, Dec. 8, 1999. See www.dmso.mil/public/library/projects/hla/guidelines/fedepv15.pdf.

Helfinstine, B., Wilbert, D., Torpey, M., & Civinskas, W. (2001). Experiences with Data Distribution Management in Large-Scale Federations, *Simulation Interoperability Workshop*, 01F-SIW-032, Sept 2001.

Hung, J., Torpey, M., & Hodum, F. (2002). Performance Cost of Using Time Management Services, *Simulation Interoperability Workshop*, 02S-SIW-041, March 2002.

IEEE (1998). *IEEE Standard for Distributed Interactive Simulation - Application Protocols*, IEEE Std 1278.1A-1998.

Kuhl, F., Weatherly R., & Dahmann J. (1999). *Creating Computer Simulation Systems, An Introduction to the High Level Architecture,* Upper Saddle River, NJ: Prentice Hall.

Reilly, S., & Briggs, K., (Eds.) (1999). *Guidance, Rationale, and Interoperability Modalities for the Real-time Platform Reference Federation Object Model (RPR FOM)*, Version 1.0, DRAFT 2, 10 September 1999. Available from www.sisostds.org.

U.S. Department of Defense (1998). *High Level Architecture, Interface Specification Version 1.3*, Draft 11, 20 April 1998. See www.dmso.mil/public/library/projects/hla/specifications/main_body.pdf.

USJFCOM (2002). *Millennium Challenge 2002 Model and Simulation Federation*, August 2002.