THREADS….Tying Integration Together

Tammie F. Smiley
Trideum
Huntsville, AL
TSmiley@Trideum.com

Lawrence Rieger
US Army TRADOC ARCIC
Fort Eustis, VA
Lawrence.a.rieger.civ@mail.mil

ABSTRACT

As the Department of Defense focuses on enabling current forces for global operations; it will be applying significant simulation resources to both force developments and training. The major challenge in creating simulation federations is making sure the federation works when War fighters arrive to use it. The steadily increasing focus on joint force representation within simulation events, both training and experimentation, means that the simulation federations supporting those events grow constantly more complex. As ever more complex simulation federations support joint force development experiments and mission-readiness exercises, the standardization of integration tests permit easier and more detailed checks to ensure diverse simulation federations will work together. Army Capabilities Integration Center (ARCIC) provides TRADOC battle laboratories with the simulation infrastructure for such experiments, and has developed a rigorous program of federation integration to ensure their simulation infrastructure properly supports the experiment objectives. Throughout the last five years, ARCIC has developed phased thread tests to track and measure technical integration of the federation architecture. The use of these innovative thread tests and supporting integration processes have enabled streamlined testing efforts, better utilizing the resources of participating battle laboratories. Thread tests are used throughout the various phases of the integration cycle. Threads can be event and non-event specific, and are archived for future use. In fact, tests threads do not have to be simulation specific, but can be used by any simulation federation, whether training or combat development events. Test threads are viewed as living documents which are continuously improved by gathering feedback, and reviewed by governing technical authorities. The paper reviews how thread tests are used throughout the community, describes the innovative work processes developed for thread-based integration activities, and describes the lessons learned over four years of increasingly complex federation integration. The processes discussed in the paper are non-proprietary and the threads themselves are readily available to the Simulation Community of Practice.

ABOUT THE AUTHORS

Tammie F. Smiley is a Senior Systems Engineer for Trideum, Inc. Tammie's current responsibilities are supporting the US Army Research Development Engineering Center (RDECOM), Communication Electronic Research Development Engineering Command (CERDEC), Night Vision and Electronic Sensors Directorate (NVESD). Tammie's previous years of Modeling and Simulation experience includes supporting Army Capabilities Integration Center (ARCIC) Joint Army Modeling and Simulation (JAMSD) in the Technical Operation Support Section (TOSS) and Event Support Section (ESS) of a portion of the Battle Lab Collaborative Simulation Environment (BLCSE), working for the Directorate of Simulation - Forward in Grafenwoehr, Germany, the National Simulation Center in Fort Leavenworth, KS and the Defense Threat Reduction Agency in Washington, DC. Tammie graduate from Prairie View A&M University with a Bachelors of Business Administration degree and received her Master of Arts degree from Webster University. She received her CMSP in 2009.

Lawrence A. Rieger is the Technical Configuration Manager for the Army Battle Lab Collaborative Simulation Environment (BLCSE). He received a BA from Belmont Abbey College in 1976 and an MS from Troy State University in 1982. He is also a graduate of the Army Command and General Staff College and the Army Management Staff College. Following active and reserve commissioned service with both light and mechanized forces, he has spent the last 27 years in the management and development of simulations for training and combat developments, working in live, virtual, and constructive environments. His prior assignment was technical deputy to the TRADOC Project Manager – OneSAF. He received the CMSP in 2008.

THREADS….Tying Integration Together

Tammie F. Smiley
Trideum
Huntsville, AL
TSmiley@Trideum.com

Lawrence Rieger
US Army TRADOC ARCIC
Fort Eustis, VA
Lawrence.a.rieger.civ@mail.mil

Introduction

The Army Capabilities Integration Center (ARCIC) /Joint & Army Modeling and Simulation Division (JAMSD) in Fort Eustis, Virginia has been the simulation technical lead for experimentation for TRADOC for the past five years. In that role, JAMSD has the responsibility for the technical integration of large scale distributed experimentation simulation events. One of the major challenges in establishing and integrating these large scale distributed events is ensuring that the underlying simulation federations will actually work when the role players and simulation inter-actors show up on start day. Software testing is not unique to the military or commercial venues and many procedures have been documented on proper testing techniques. What is different in the Military simulation environment, and those industry environments which support the Military environment, is the scope and type of software testing for a federation of simulations. What is unique in this environment is that software functional testing is really not the focus; each piece of software is already assumed to work before integration starts. It is the "Plays Well with Others" software testing that is the primary focus of federation integration testing.

Thread Testing Defined

In basic definition, a thread is an end to end process to prove a postulate, essentially saying, if this process goes from start to finish, I know that X will result. Within simulations, we generally are looking for one of two results, either the process works, or doesn't; and either the end result meets this outcome, or it doesn't. Our two thread tests are to ensure the process works and to ensure the working process provides the expected value when completed. It is the second test, the outcome with data values, which makes thread tests in the simulation environment different from most thread tests used in computer software, mechanical engineering, or even automotive maintenance. In those fields, the threads are primarily process. In simulations, especially in distributed simulation federations, the data outcome thread, also called the validation thread, is as important, if not more important, than the process thread.

As basic examples, a process thread is like starting your car. Given a car, key, connected and charged battery, and all pieces and parts properly connected, the turning of the key in the ignition will cause switches to turn, power to flow, and processes to function; and the test execution results in an operating motor. The test developer can define each of the thread steps that occur between inserting the key and the motor starting. A validation thread however, measures the quality of the result of the process rather than the completion of the process. If each truck in a given convoy has a one percent chance of a flat tire at each turn, and a five percent chance of making a wrong turn, and the operation begins with one hundred trucks in the convoy and ten turns in the road, the quality of the end of the process is expected to be about forty trucks in the convoy at the end point. When these thread tests are run, if the truck engine didn't start, or if there are only ten trucks at the end point, tracing the sequential steps within the end to end thread enables the tester to determine what failed, and where.

Federation testing levels and processes

Federation integration testing actually occurs on three levels. First there is the simulation functionality as a federation. There is also the data set interaction functionality at a fair fight level of interactivity. And then there is the Toolkit stimulation functionality to meet the objectives of the event, primarily generation of recordable empirical data and the stimulation of Mission Command (Battle Command) systems. If it is a training federation, this part also includes stimulating the participants to react as the training program plans.

Each of these three levels will be discussed in detail. But the starting point is the deliberate set of test plans and procedures that give the simulation event manager the ***assurance*** that the federation works when the expensive start button is pushed. These test plans and procedures, which JAMSD calls thread tests, are both documents in themselves, and a measurable road map for the event manager to follow. JAMSD views these testing threads as a conceptual testing tool relating a contiguous set of steps intended to evaluate a given capability fulfilling a federation requirement. Writing threads in the Military simulation environment is not as detailed as it is at the software development level but it still needs to be handled on a similar scale. For example, the developer will verify that basic demands of the program are functional such as input and output of the system and then will give it a green light to move up to the next level in their organization. This could be database development which will run similar tests for basic functionality or to the alpha release of the product to internal teams. Test threads bridge the communication gap between developers, operators and the Federation Managers or Battle Masters, which allow capabilities to be analyzed to mitigate the risk of the experiment or software.

“Threads not only serve as the basis for integration, they also tend to drive the entire software development effort from scheduling to status reporting. Each thread itself represents a microcosm of the system in that each has a documented definition and general execution path, an internal design and an associated test." (Ahamad, 2005)

This paper highlights and describes innovative work processes that were developed to support effective and efficient thread-base integration activities.

Thread Process and Procedures used in the BLCSE Environment.

The Battle Lab Collaborative Simulation Environment (BLCSE) can be defined as a complex distributed “toolbox” capable of supporting multiple models and human in the loop simulations. Together with the network infrastructure to connect the various sites, it enables the distributed experiment planning, execution and analysis essential to the Army Capabilities Integration Center (ARCIC). Within the BLCSE, ARCIC JAMSD actively plans and manages the integration of a large and heterogeneous federation of simulations which reside at multiple locations and each of which bring different functional or technical capabilities to the federation. Our typical federation would consist of One Semi-Automated Forces (OneSAF) to represent most ground force entities, FireSim to represent the precision processes and effects of indirect fire support, Advance Tactical Combat Model (ATCOM) to represent rotor wing aircraft and unmanned aerial sensors (UAS), Air Warfare Simulation (AWSIM) to represent joint fixed wing aircraft and aerial combat, and Extended Air Defense Simulation (EADSIM) to represent spaced-based sensors and often naval vessels. As these simulations have different internal processes, different earths models and often use different Federation Object Models (FOMs); the integration process to a state of plays-well-together is prolonged and costly. As the various simulations are owned and controlled by different commands and centers, or different services, and as there are often conflicting schedules and mission, the heart of the integration mission is developing a process that provides entrance and exit criteria in specified phases of integration. Thread testing enables the various proponents to know what they need to enter each integration phase, and then what they need to exit each integration phase, and prevent mandatory attendance in integration activities when they are not necessary to the execution of the threads for that day. Going back to the issue that the various simulations are owned, and resourced, by different commands, centers and services, an integration process that enables a reduction in schedule time, manpower resources, and number of integration days, are major advantages. In a heterogeneous federation, the recommended practice is three phases. Individual interoperability with the standard as Phase I, with entrance criteria a working simulation and exit criteria being assurance of enumerations mapping to the baseline. Collective interoperability as a federation as Phase II, with successful exit from Phase I being entrance criteria and exit criteria being assurance of cross-federation enumerations (data mapping) across the RTI including the federation tools. Federation functionality under stress as Phase III with successful Phase II being entrance criteria and exit criteria being a full load (all entities and federates operating) data interoperability with the face validation of outcome results being appropriate to the simulation events.

Defining the operational environment

Within BLCSE events, JAMSD functions as the integration lead, and as such, performs three primary integration functions. As overall technical lead, they are responsible for the planning and scheduling of integration activities, and directing the integration activities. As software configuration control authority, a very necessary function of integration, it ensures that threads are appropriately scheduled based on the release of various software and dataset versions. And as integration functional lead, JAMSD provides the federation control and simulation integration management, aka “Battle Master” who is critically important in applying the thread testing in the integration environment.

In the technical lead function, JAMSD divides the integration activities into three phases. Phase I ensures that each of the various simulations interoperates with a common simulation denominator (OneSAF) and tests entity type enumerations to prove the Federation Object Model (FOM). Phase I is almost exclusively process threads. Phase II ensures that all of the various simulations interact fairly within the federation as a whole, including specialized tools such as a damage effects server. This phase has predominately hybrid process and validation threads. Phase III provides assurance that the federation as a whole works with all tools, battle command systems, and datasets; and is also predominately hybrid process and validation threads. The thread scheduling by phase becomes a demanding management process, and is described below in detail. The technical process of configuration management and control as an integral part on integration scheduling has been previously presented to the community. (Martin & Rieger, 2010)

Define the Thread Test Need

The greatest challenge in developing threads for integration comes in requirements determination. While software development process threads can tend to be very straightforward, and are built from a specific system engineering design document, the same is not true of federation integration. Standard software development thread testing is essentially ensuring that a system processes data. Federation simulation integration thread testing assures that a system of systems processes data for an appropriate end result. Therein lies the major difficulty in designing and developing the integration threads. The Integrator must know the use cases and the federation application, which well may change from event to event. This challenge is made difficult in that, unlike the development of a system, there is usually not a requirements document or a systems engineering diagram for the simulation event. Here, what becomes the Integrator's primary tool is the event design based on using the Department of Defense Architecture Framework templates. (DoDAF, 2012) The authors recommend a minimum set of the Operational View 1a (OV-1a), the Operational View 2b (OV-2b) and the Systems View 1 (SV-1) in order to graphically depict the flow of data between the various simulations and tools used within a federation. The OV-1a, shown at Figure 1, provides the best overall graphic depiction of the data flow necessary for threads development. Both of these additional toolsets, Battle Command and data collection systems must co-exist with the simulation federation in order to meet the analytical objectives of the event. A training event would require the same type of architecture diagram, but would have the advantage of less change between events.

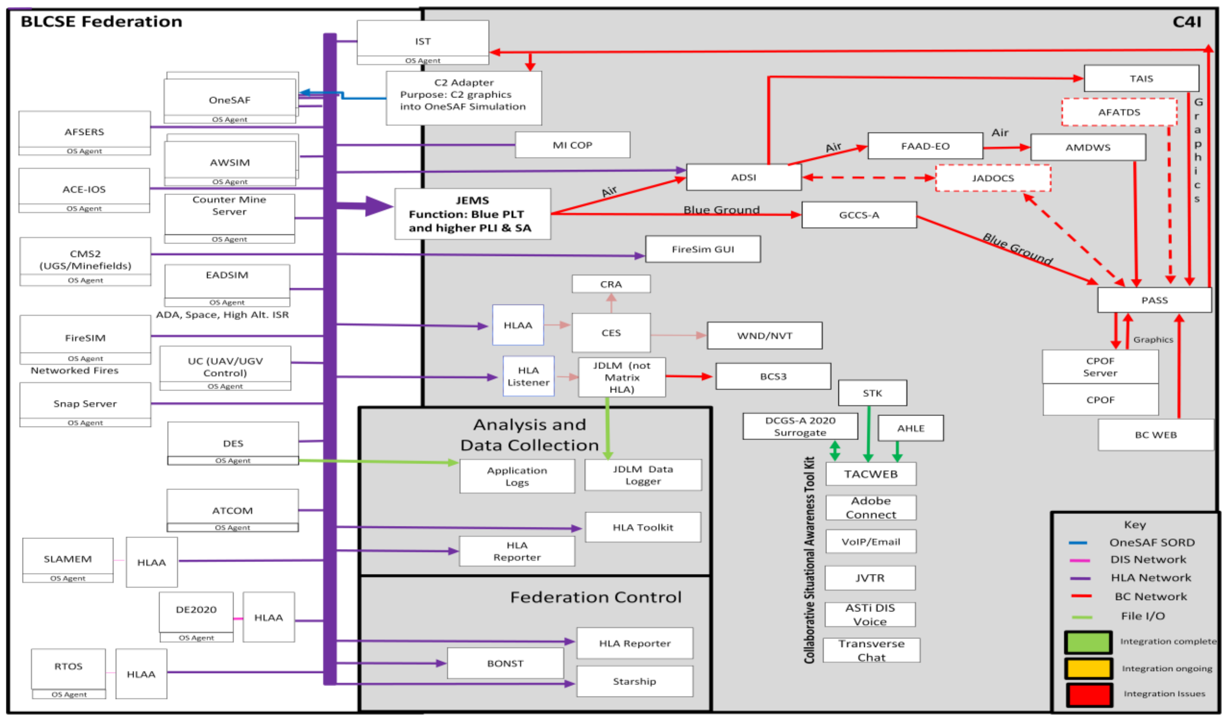


Figure 1: Operational View -1a (OV-1a)

The BLCSE experience has shown that use of the OV-1, or its equivalent, is necessary for effective and efficient mapping of threads, particularly process threads, in assuring the architectural functioning of a federation design.

Having in hand the production schedule of the supporting datasets (parametric, terrain and player) and the event schedule, the Integrator then backward plans their integration phases tied to the delivery of products and the configuration management plan. The authors note that in today's continuously evolving, and very demanding, Information Assurance (IA) environment under the DOD Information Assurance Certification and Accreditation Process (DIACAP), scheduling the configuration change of dataset and software patches with the fault detection purposes of thread tests becomes a major administrative headache. The thread test plan has to be developed with the knowledge that thread tests will find data and software errors, which will have to be fixed under Configuration Control Board oversight, and then those changes themselves will need to be checked for functionality. The authors strongly recommend that the facility Information Assurance Manager (IAM) be involved in the schedule planning for both software version, patch and dataset releases and their associate thread plans to enable both CM and IA to be accomplished within the integration activity.

Development of the Test Threads

In the development of threads for integration, the first question asked by the Integrator is, what do I have to have ***assurance*** of ***before*** the event starts? The usual response is, of course, everything; but that begs the question of what is the purpose of the event. There will usually be a central purpose for the event which drives the thread testing focus of the Integrator. In a training event it may be the stimulation of the Battle Command systems by the simulation suite. In an experimental event, it may be determining the best number of a type of sensor that a particular unit needs for a future battle. The focus will drive the concentration of threads in the later part of integration, which weights the number and type of threads in the earlier part of integration. The authors recommend that the integration and its threads are aimed, and echeloned, at answering these four questions in priority order.

<u>Integration Thread Questions</u>

1. Do each of the simulation federates exchange data with the FOM using the enumeration schema?
2. Does the federation, as a data construct, have full end to end functionality between the simulations and the toolsets?
3. Does the federation as a whole provide the stimuli and data outputs necessary to accomplish the objectives of the event?
4. Does the federation as a physical construct, including the communications backbone, have the stability to sustain full data flow between federates?

In the BLCSE environment, the development and management of test threads is an integral part of our Model and Simulation Support Plan (MSSP) and our event integration process. Within the M&SSP, a document updated weekly during active integration, an execution matrix outlines the threads scheduled for the following week. The threads selected are based on the master thread test list, and the integration focus of that week of integration.

Beginning with the long lead time Concept Development Conference (CDC) and Initial Planning Conference (IPC), the event architecture is determined between the event manager, the event technical integrator, and the various participating sites. As mentioned earlier, this architecture is documented using DoDAF, with particular focus on the OV-1a, OV-2b and SV-1 schema. These three graphic depictions are sufficient for the integration team to develop an initial listing of all threads that are required for the event being integrated. It is worthwhile to note that the first time an event is being set up as a new federation, there is a very long and painstaking period of systems engineering and federation analysis to develop the initial thread test menu. JAMSD's first methodical implementation of thread test management required nearly six months of work by two engineers to study the DoDAF for the BLCSE federation, evaluate the simulation and battle command system data flows, and determine all of the end to end and data validation threads that were required. And after that thread test listing was completed, one engineer remained full time in direct support of threads development, modification and documentation, completely separate from the Battle Master management of the actual thread testing being conducted by the federation. The bottom line is that, even starting from a level of some experience and with institutional knowledge of previous integration processes, establishing a comprehensive thread test index for a complex federation of heterogeneous simulations, tools and battle command systems is between one and a half to two man years of effort. Once established however, minimal effort is required to sustain the list of threads as a federation index, and most of that activity falls on the federation battle master (or integration controller) as part of the integration management process. Within the most recent Gain and Maintain Operational Access (GAMOA) experiment, thread test maintenance required only about three engineer-weeks of effort, about the same amount of time required to plan and schedule the threads.

The below list of ten key tasks and procedure steps represents the system engineering, thread test development, validation, configuration management and thread implementation required for the first comprehensive federation integration using a methodical thread test index. Although BLCSE has since streamlined the technical teams, the initial expedition through thread development required substantial interaction between the Federation Management Support Section (FMSS) who did simulation software support; the Technical Operation Support Section (TOSS) which managed technical integration; and the Event Support Section (ESS) which performed integration management and configuration management. In generic terms, FMSS were the simulation programmers and data load folks, TOSS managed and integrated, while ESS planned, prepared and configuration managed the processes and documents. The authors strongly recommend that the first time any activity establishes and integrates a heterogeneous federation, e.g., a cold start, equivalent structure and processes be executed to develop the integration knowledge and procedures.

1. **Creation of thread (ESS):** The ESS gathers the technical details based on the Operational Order (OPORD) requirement document of the event and a prototype architecture schema. If OV-1 is delayed in planning, past experiences are used to generate and recommend threads that should be used. In addition, JAMSD contacts the simulation proponents and event participants to request their operational and technical requirement for the event. This is particularly important for any new federate or capability that needs to be tested to ensure interoperability within the federation.

2. **Conduct Alpha Testing of thread (ESS):** To validate products, ESS has an internal review and execution. Where possible, conduct internal thread development testing for this procedure.

3. **Meet with Battle Master on upcoming threads (week prior) (ESS/FMSS/TOSS):** This meeting is based on a question and answer session. The Battle Master (BM) can ask for further explanation on test threads that have been developed. In addition, the BM can request additional test threads for the upcoming integration test week based on results to date and interaction with the various federation members.

4. **Place threads under Configuration Management (CM) using the Dimension Server:** Configuration control allows other thread developers and or testers to collaborate on modifications within JAMSD policies. CM also permits the reuse of threads between events.

5. **Add Threads to the Thread Index/Test Plan (ESS):** The thread index provides a listing of the thread name and the brief description. Weekly Test Plan lists only test threads, which will be utilized to test against specific testing objectives of a given week.

6. **Deliver threads to Battle Master for execution (ESS):** Ideally one to two weeks before execution; NLT Thursday before the execution week (Note: Thursday is M&SSP publication day in JAMSD practice)

7. **Battle Master provides feedback to ESS:** During daily “hot washes” the threads are updated to the thread development engineer, ensuring threads adequately represent and evaluate the process and validation flow necessary to successfully validate the test item.

8. **Upon receipt of the feedback the ESS rep will review and/or make modification as needed:** All changes are confirmed with the logical data flow and software functionalities through FMSS and changed and updated threads are entered into configuration management.

9. **ESS provides a daily status report to the Battle Master:** Ensuring that the Battle Master has the appropriate or updated thread test documentation for upcoming integration testing.

10. **Once issues are resolved; the thread will be re-tested at the discretion of the Battle Master.**

In keeping with the goals of development and testing, each thread has its own lifecycle with states and a formal process for state transitions. Experience demonstrates that, first time through, extensive coordination, and to some degree, spiral development, is necessary between the thread developers, the software test engineers, and the simulation developers who had to make fixes and changes in their simulations or tools to enable the federation as a whole to function.

After the first full cycle of federation integration, and the development of the thread index with mapping to the various integration phases, JAMSD was able to significantly reduce the operational complexity of the thread test and integration process, as the initial development cycle was found to be "over-engineered."

Categories of Threads

Throughout the integration process, a test thread has its own lifecycle depending on its capability and re-usability. As previously discussed, threads are functionally divided between process and validation types. But within these functional groupings, there are four categories of threads to define and discuss: specific, non-specific, technical, and operational threads.

Specific threads are designed to test event specific capabilities. This could be a certain requirement, for example having the Navy play in an Army-centric federation conducting amphibious assaults. This thread's lifecycle usually retires after the exercise is completed; however, their generated feedback can stretch to other exercises and influence future events.

Non-specific threads are designed to test general features of the system that don't require specific devices and can easily be used in future exercises. For example, one may choose to test for a "I see you, you see me" thread, which verifies the mapping of entities across a DIS or HLA federation.

Technical – Technical threads are used to test the technical features of the simulation. This is usually something that is not seen by the operator but is a requirement to run a successful operation. For example, "checkpoint save and restore", is a feature of the program that allows either manual or automatic snapshot of the exercise and restores based on the time it was taken. This technical capability thread documents exact steps on how to initiate a checkpoint save and restore and compare the before and after results. This would be a specific thread by device but its lifecycle would be endless based on the software feature.

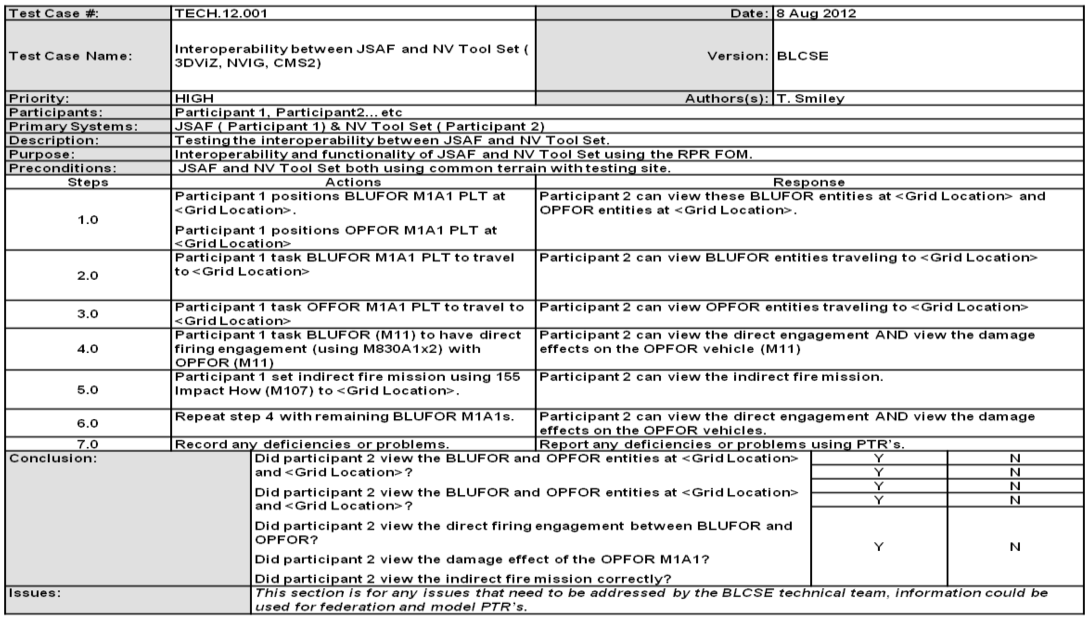
Operational – This thread is the most common due to the need to test certain conditions for the exercise to be successful. Testing the capability of sending message traffic to and from a constructive simulation to a C4I device is an example. Once again this could be federate specific but are usually found in all exercises that have cross-level federations: Live, Virtual, and Constructive environments.

During the 2011 Joint Forced Entry War fighting Experiment (JFEWE), JAMSD had specific details in the threads that called out what system, site and sometimes entities needed to execute. However, the community of practice found these threads to be restrictive and at times unable to capture their requirements. Therefore, for the following event, Wide Area Security War fighter Experiment (WASWE), thread reform was introduced and implemented. Now, the threads are developed with an "off the shelf" ("OTS") reusability concept. This "OTS" concept proved its flexibility during the 2012 GAMOA event when very dynamic event managerial changes occurred. Numerous late event objective, scenario and participant changes were experienced and the event was refocused. The resulting technical architecture and entity player changes demanded thread scheduling flexibility literally on a weekly basis. The major lesson learned from this event is to make the threads detailed enough to understand what needs to be accomplished, but not as detailed as a spaceship launch checklist.

Integration Phases

Let's take an in-depth look at each integration phase. Throughout all phases of technical integration, threads are an important part of testing. Each phase represents a stepping stone leading up to the execution of the experimentation event or War fighting exercise.

Pre-Integration (Phase I) – This phase answers Integration Thread Question one. During the pre-integration, one-to-one federate system testing is conducted. Q: Can federate A view, interact and engage (sometimes) with federate B and vice versa?



Test Case #:	TECH.12.001	Date:	8 Aug 2012
Test Case Name:	Interoperability between JSAF and NV Tool Set (3DViZ, NVIG, CMS2)	Version:	BLCSE
Priority:	HIGH	Authors(s):	T. Smiley
Participants:	Participant 1, Participant2... etc		
Primary Systems:	JSAF (Participant 1) & NV Tool Set (Participant 2)		
Description:	Testing the interoperability between JSAF and NV Tool Set.		
Purpose:	Interoperability and functionality of JSAF and NV Tool Set using the RPR FOM.		
Preconditions:	JSAF and NV Tool Set both using common terrain with testing site.		
Steps	Actions	Response	
1.0	Participant 1 positions BLUFOR M1A1 PLT at <Grid Location>. Participant 1 positions OPFOR M1A1 PLT at <Grid Location>	Participant 2 can view these BLUFOR entities at <Grid Location> and OPFOR entities at <Grid Location>.	
2.0	Participant 1 task BLUFOR M1A1 PLT to travel to <Grid Location>	Participant 2 can view BLUFOR entities traveling to <Grid Location>	
3.0	Participant 1 task OFFOR M1A1 PLT to travel to <Grid Location>	Participant 2 can view OPFOR entities traveling to <Grid Location>	
4.0	Participant 1 task BLUFOR (M11) to have direct firing engagement (using M830A1x2) with OPFOR (M11)	Participant 2 can view the direct engagement AND view the damage effects on the OPFOR vehicle (M11)	
5.0	Participant 1 set indirect fire mission using 155 Impact How (M107) to <Grid Location>.	Participant 2 can view the indirect fire mission.	
6.0	Repeat step 4 with remaining BLUFOR M1A1s.	Participant 2 can view the direct engagement AND view the damage effects on the OPFOR vehicles.	
7.0	Record any deficiencies or problems.	Report any deficiencies or problems using PTR's.	
Conclusion:	Did participant 2 view the BLUFOR and OPFOR entities at <Grid Location> and <Grid Location>?	Y	N
	Did participant 2 view the BLUFOR and OPFOR entities at <Grid Location> and <Grid Location>?	Y	N
	Did participant 2 view the direct firing engagement between BLUFOR and OPFOR?	Y	N
	Did participant 2 view the damage effect of the OPFOR M1A1?	Y	N
	Did participant 2 view the indirect fire mission correctly?	Y	N
Issues:	*This section is for any issues that need to be addressed by the BLCSE technical team, information could be used for federation and model PTR's.*		

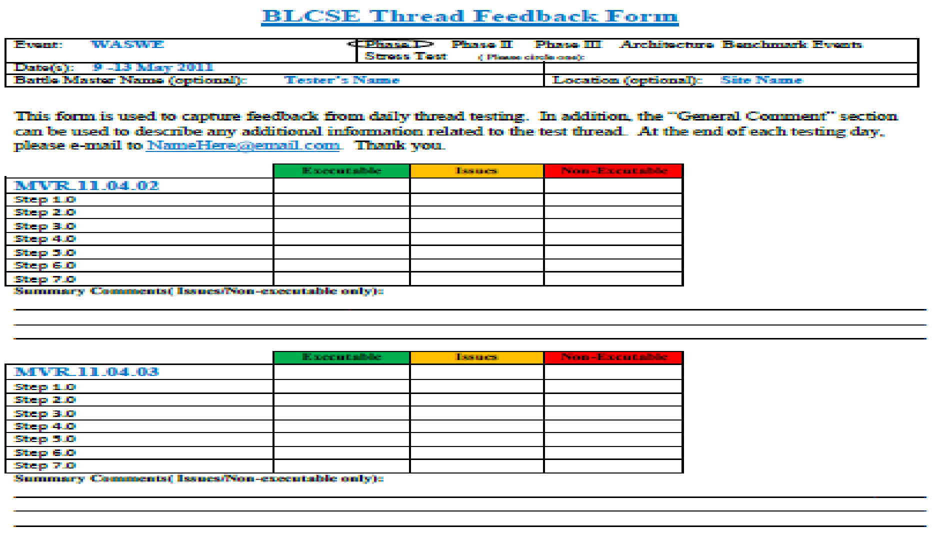
Figure 2: Interoperability Test Thread

For example, testing the basic functionality and insuring entities are transmitted across the federation correctly (see *Figure 2)*. In addition, identifying if any interoperability issues exist. One of the most common issues during this phase is mapping issues.

Integration (Phase II) – This phase answers Integration Thread Questions two and three. One-to-many testing is conducted during this phase. Federate A is conducting testing with Federate B, Federate C, and Federate D, etc. Therefore, this phase is usually the most involved out of the three phases. Vignettes and threads are used to validate basic simulation events. Most problems are identified during this phase, and the majority of the threads are reporting threads and interaction threads. These threads also require validation and are comprised of more specific thread types. Depending on the level of the audience the thread might need detailed outlines on how to have a feature work which would require contacting the Subject Matter Experts (SME).

Post-Integration/Pre-event (Phase III) – This phase answers Integration Thread Question four. The final phase of test threads collectively integrates all systems (many-to-many). These threads are highly specific to the exercise and are based on the Master Scenario Event List (MSEL). For example, a Vehicle Borne Improvised Explosive Device (VBIED) is detected at a checkpoint and detonates. The reactions are reflected through the simulation to stimulation of the C4I devices. The results in these exercises should be shared with all those involved with the exercise to validate if the learning objective is being met.

Feedback – Test threads are living documents and feedback allows the thread developers to validate risk assessments that could be a potential "red light" for the experimentation event. The Integrator has to know, from the technical integration leads for each simulation and site, that the threads adequately provide event readiness assurance. Since this requires documentation from many levels, at times it is the hardest to realize. We have realized success in this using two methods; capturing real-time data through the use of the Battle-Master log, and also through using a basic thread test form similar to *Figure 3*, located on page 9. Regardless of method format, feedback is required during all phases of integration. This feedback is the single most valuable driver in ensuring that the threads library, when matched to the architecture horse blanket, will provide assurance to the event Integrator. It is also key to thread reuse in future events, ensuring the thread provides the detail required for success, matched to a current federation.



BLCSE Thread Feedback Form

Event: WASWE	[✓ Phase I] Phase II Phase III Architecture Benchmark Events Stress Test (Please circle one):
Date(s): 9 -13 May 2011	
Battle Master Name (optional): Tester's Name	Location (optional): Site Name

This form is used to capture feedback from daily thread testing. In addition, the "General Comment" section can be used to describe any additional information related to the test thread. At the end of each testing day, please e-mail to NameHere@email.com. Thank you.

	Executable	Issues	Non-Executable
MVR.11.04.02			
Step 1.0			
Step 2.0			
Step 3.0			
Step 4.0			
Step 5.0			
Step 6.0			
Step 7.0			

Summary Comments(Issues/Non-executable only):

	Executable	Issues	Non-Executable
MVR.11.04.03			
Step 1.0			
Step 2.0			
Step 3.0			
Step 4.0			
Step 5.0			
Step 6.0			
Step 7.0			

Summary Comments(Issues/Non-executable only):

Figure 3: Basic Feedback Form

Thread Maintenance

Utilizing feedback and lessons learned are key improvement areas to support the thread maintenance. Keeping the threads updated based on the feedback is essential for software enhancements and preparation for the next event. Simulation and federation tool Problem Trouble Reports (PTRs) are generated based on an issue which results from the thread test. For example, during execution of the checkpoint/restore test thread – the restore function within the constructive simulation was not working as expected. Therefore, a PTR was submitted to fix this issue. Once this issue is resolved, the thread generally must be re-tested and functionality validated.

Using matrixes

Although the great majority of integration testing can be appropriately assured through the use of thread tests as linear checks, there are three areas of integration testing that are not well suited for this type of checks. The three areas are entity enumeration cross-checking (mapping), kinetic effects interactions (gunfights), and sensor acquisitions (detections). As the various simulations are integrated, each of them traditionally uses their own internal mappings of entity types and IDs, even when the federation as a whole has agreed on DIS enumerations or HLA FOM enumerations. There are an enormous number of entities, from individual soldiers and weapons through aircraft and naval ships, which must be mapped across the federation so that all simulations see the appropriate entity when interacting. For example, when Simulation A generates entity information about soldier B, does Simulation C see soldier B, or does it see Tank D? Once the federation has agreed on the common federation master enumeration mapping, the Integrator has to make sure that mapping is the same across the federation. Standard thread testing is ineffective in this application, so matrix testing is created.

JAMSD developed a set of threads specific to this issue, called ICU/UCMe (I see You, You See Me). These involve generating a "One Each" scenario file for each federate, and building a simple spreadsheet consisting of the one-each list on one axis and the various simulations, or tools, on the other. Placing the One-Each scenario file on the simulation terrain, each simulation sends its one-eaches in parade order past the one-eaches of another federate, in a known order. The Integrator and various simulation proponents then check off that each federate says, yes, I am generating this entity, and yes, I am seeing that entity. As the Integrator/Battle Master works through the parade scenario, he is able to assure that each simulation does in fact have the entity properly mapped, both transmitting and receiving (generating and perceiving) , and can keep track of that assurance through the matrix chart.

The same process is used for munitions as for entities, with an additional facet in that the value of the engagement outcome is also checked. For example, there might be two hundred separate entity types in the federation, again ranging from a dismounted infantryman through a main battle tank to a naval warship. Each munition is separately mapped into a matrix axis, with entities being listed on the other axis. Each munitions used by each entity is checked to see that it can be fired at another entity, the fire interaction confirmed, and then looked at to determine if the fire interaction provided an appropriate kinetic effect. Did the 5.56mm munitions affect the infantryman? Yes=good. Did the 5.56mm munitions affect the tank? Yes=bad.

BLCSE developed and maintains a comprehensive ICU/UCMe matrix for enumeration mappings, and a corresponding IShootU/UShootMe matrix for kinetic effects. The same munitions type matrix thread is used for sensor acquisition and detections as for munitions, except the quality aspect is acquisition at range and condition rather than kinetic result for munitions and target.

Although relatively simple in concept, a two axis spreadsheet; most matrices are more detailed in application in that they would have entity vs. entity, observer vs. entity in various terrains, or any other combination for which the system is being tested. Such a matrix and thread would be run many times during a testing period to check and assure that the results actually match realistic expectations.

What is a very important lesson-learned in matrix threads is that the total body of entities, munitions and sensors has to be identified very early in the event development cycle, and that, if you are adding new parametric data to simulations during the integration process, these matrix threads have to be appropriately scheduled.

JEFWE 2011

In the 2011 JFEWE event, JAMSD developed limited detail threads which worked for the majority of the test cases but not all. The Navy joined the usually Army-centric exercise and required amphibious landing and assault. Therefore, this capability was tested throughout the Phase II and Phase III schedule periods of integration. What showed up during these phases were unmapped entities and munitions, as new functionalities and new vehicles and weapons were added. Not being part of the initial architecture, both enumeration and functionality integration testing was needed to test the capability as well as the interoperability effect of the federation. Based on the testing feedback the event did have minor issues; however, the issues were resolved before actual event record runs. Also added as a test was Improvised Explosive Device (IED) capability at a checkpoint. During execution of this capability a major issue was identified. Therefore, the exercise controller was informed and simulation and dataset adjustments were made. These two examples are detailed threads for a particular event that were not in the re-use library. These threads have a onetime use life unless this capability is needed for another event. The authors recommend that “one-time use” threads still be retained in an archive thread index for use in later events.

Another aspect of testing during JEFWE, was having threads that specified federate systems that were needed to execute that particular test. This was an issue because at times the federate was not available and this caused a disruption in our testing schedule. One of the lessons learned from this event was to tailor from detailed threads (federate and site specific) to more generic centric. Again this goes back to the idea that a thread test document needs to be much more descriptive rather than prescriptive.

Current Adaptations

As a result of the lessons learned in 2010 and 2011 simulation events and the benefits gained from having established and entered into configuration management the body of BLCSE thread tests, JAMSD is now able to operate in a much more streamlined, and less rigid, process. Rather than having a dedicated threads engineer, and a separate experiment planner, the threads execution matrix is managed by the federation Battle Master as part of the integration schedule, being modified almost on the fly to meet the needs of the integration schedule. As noted before, the reuse and maturity of the threads process has reduced resource requirements from over a man-year to less than two man-months.

BLCSE thread management has been moved from a large and detailed separate plan into a one-week look ahead execution matrix published weekly as a tab to the Model and Simulation Support Plan. The authors note that Thread Tests are now firmly integrated into various plans of the Distributed Simulation Engineering and Execution Process (DSEEP) 2010industry standard, specifically the Test Plan, Configuration Management Plan, Data Management Plan, and the Integration Plan. (DSEEP, 2010) The use of the DSEEP plans, integrated with threads in the appropriate documents, is recommended as an integration best business practice, and enables separate thread test plans and schedules to be eliminated as standalone documents. Rather, the OV-

1-based thread plan becomes integral to the various DSEEP plans and integration schedule.

In summary, the authors have found that the use of deliberate threads, both process and validation, have enabled BLCSE to shift the more rigid and lengthy BLCSE E-Date (Execution Date) sequence from a seventeen week block to a fifteen week block. The integration schedule has re-aligned the focus from three, equal, four-week integration phases to a three week Phase I primarily with process threads, a six week primarily matrix thread Phase II, and a three week Phase III focused on event validation threads. As the GAMOA event ends in June 2012, the technical review for the integration engineering staff will be formally evaluating a substantial decrease in the total length of the integration period, enabled by disciplined thread testing enabling a much more efficient federation integration process.

Conclusion

The JAMSD testing and integration process has evolved over the past five years. Each completed event generated lessons learned to continually refine the integration testing process. Thread-based integration testing plays the key role in the technical success of a distributed simulation event. Testing the operation and technical requirements during the integration period is a strong management tool for event risk mitigation. The strength of the use of threads is not just in their ability to use a disciplined systems engineering testing process, but also, through the use of a threads management matrix chart, it provides the event manager with an integration engineering metric tool for the evaluation of progress and for the scheduling of resources. The event manager can use thread testing as a means to assign integration engineering resources where they are most needed to reduce technical risk to the important pieces of the simulation infrastructure. This allows for software enhancements before execution and the evaluation of technical risk both before and during the simulation event. In addition, the various levels of testing require an integrator to focus on testing capabilities across the federation. One important mechanism – Threads… ties it all together!

ACKNOWLEDGEMENTS

We would like to express special thanks to Ann O'Kennon for her editorial contribution for this paper.

REFERNCES

Ahamad, K. (2005). *Thread Based Integration Testing*. Retrieved 20 May 2012, from http://testingsoftware.blogspot.com

Martin, Kimberly and Rieger, Lawrence (2010). *Configuration Control in a Cross-Distributed Team Environment- Preventing the Tower of Babel.* IITSEC 2010 Proceedings.

Department of Defense Architecture Framework. (2012). Retrieved 21 May 2012, from http://en.wikipedia.org/wiki/Department_of_Defense_Architecture_Framework.html .

1730-2010 IEEE Recommended Practice for Distrubuted Simulation Engineering and Execution Process(DSEEP). (2010). Retrieved from http://standards.iee.org/findstds/standard/1730-2010.html