Configuration Control in a Cross-Distributed Team Environment - Preventing the Tower of Babel

Kymberly Martin
Reger
Abilene, TX
kym.martin@regermail.com

Lawrence Rieger
Army Capabilities Integration Center
Joint & Army Models & Simulations Division
Yorktown, VA
lawrence.rieger@us.army.mil

ABSTRACT

The Battle Lab Collaborative Simulation Environment (BLCSE) is a complex consortium of military, government civilians, and contractors tasked with integrating simulation technologies and other supporting applications across geographically distributed sites. A key part of this integration is the interoperability of data between simulations. As projects, missions and teams grow in scope greater data interoperability challenges ensue. Therefore, a clear method to configuration manage and control data sets, their changes and related documentation and other files is necessary. Project management alone isn't enough to ensure experiment success, you must have a data and software configuration management process. Configuration Management (CM) is a method that provides structure to a project. Implementing a configuration managed environment forces discipline where otherwise disorder will ensue. An agreement among participants is paramount to success. The mechanism to provide control is a combination of CM and collaborative tools. This paper describes a standard and documented process for data and software configuration management necessary for terrain, entity data, HLA interoperability files and related configurable items. It also describes lessons learned in establishing a vigorous scheme of configuration management during a major distributed event, compares tools and processes available, and concludes with a discussion of configurable items appropriate to a major distributed simulation event. Armed with the right tools and processes, a simulation management team can properly manage data changes so that the federation enables training and experimentation, and the federation users can train to fight and fight to win.

Kymberly Martin has a BS in Electrical Engineering from Kansas State University. She is a member of the IEEE and the Society of Women Engineers where she is a member of the emerging leader awards committee. She has 15 years experience in systems engineering and configuration management. She is currently the BLCSE configuration management engineer for Reger Associates, and has previously worked configuration management for the B-1 Bomber program working for Rockwell Collins and was a system engineering test team member for GPS integration at Holloman AFB. She received her CMSP certification 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 25 years in the management and development of simulations for training, 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.

.

Configuration Control in a Cross-Distributed Team Environment - Preventing the Tower of Babel

Kymberly Martin
Reger
Abilene, TX
kym.martin@regermail.com

Lawrence Rieger
Army Capabilities Integration Center
Joint & Army Models & Simulations Division
Yorktown, VA
lawrence.rieger@us.army.mil

INTRODUCTION

Perhaps the greatest challenge in distributed simulation events is ensuring that all of the various participating software applications can effectively exchange data, with everyone having the same coherent picture of what that data actually means. Without a common and shared agreement as to the meaning of the data, and how it is processed, the simulation federation is unable to provide a stable environment for successful completion of the events' objective. It is configuration management that provides the disciplined process and procedures to ensure common data models and processes among all the participants.

Background

In 2005, the TRADOC Futures Center (now the Army Capabilities Integration Center (ARCIC), determined the requirement to transition its simulation infrastructure (The Battle Lab Collaborative Simulation Environment – BLCSE) from the existing Distributed Interactive Simulation (DIS) standard to the High Level Architecture (HLA). The driving issue was the need of BLCSE to interoperate as part of a broad Army effort (the Cross-Command Collaborative Effort – 3CE).

Part of the decision process was the recognition that with HLA, we needed extremely tight configuration management of the FOM and enumerations, rather than being able to rely on the DIS enumerated bit values. We also recognized the need for strong CM practices over the version(s) of software applications being used within an event. The technical assessment of the ARCIC was that movement to the HLA would require very strict Federation Object Model (FOM) and cross-event entity enumeration configuration management for a successful transition. Three years of distributed experimentation events later, our greatest integration challenge, and source of simulation interoperability friction, is FOM currency and ensuring that all simulation federates are using the exact same list of entity enumerations and the exact same FOM files.

In addition to the increased Configuration Management requirements of moving into an HLA environment, ARCIC was also beginning to see initial information assurance requirements, from the then emerging DoD Information Assurance Certification and Accreditation Process **(**DIACAP) requirements, which would demand a much higher level of configuration management and configuration control than the existing DoD Information Technology Security Certification and Accreditation Process (DITSCAP).

Once the decision to move forward was made, ARCIC developed an initial Configuration Control Board (CCB) charter and draft configuration management processes to govern the transition of BLCSE to HLA. This process was complicated by the distributed environment and the practical reality that the configuration control board neither owned nor could control the various simulations that were nominally being managed for BLCSE events.

A management process was developed during this period which separated the proponency of a simulation, and the resulting proponent managed

baseline, from the event specific versions which were separately managed and controlled by ARCIC and the event technical manager. This became the basis for the BLCSE configuration management process, and is shown in Figure 1. We also developed (purchased) a set of configuration management tools which supported our processes, and developed a taxonomy for our processes.

Definitions

These five definitions provide the context for this paper:

- **Configuration Management:** The identification of the configuration of software at discrete points in time and the systematic control of changes to the identified configuration for the purpose of maintaining software integrity, traceability and accountability throughout the software life cycle.
- **Configuration Control:** After establishing a configuration, the evaluation and approval of changes to the configuration and to the interrelationships among system components.
- **Baseline:** A simulation software package that is under configuration management and in continued development or maintenance for use by customers.
- **Version (Release):** An approved snapshot of the system at appropriate points in the development life cycle.
- **Configurable Items:** Specified items (software, documents, hardware) which are content and change controlled for a specified event or period.

Configuration Management and Configuration Control

There are three variations of configuration management and control executed within BLCSE. The first is Software Baseline Configuration Management, which is the process whereby each application proponent maintains configuration management and configuration accounting over that application. This is essentially cradle to grave control and documentation of everything done to that software application during its lifespan.

The second is Event Configuration Management, where a particular application version is adapted and controlled for a specific event, after which that configuration branch is archived and no longer part of the software baseline.

The third is Event Configuration Control, which is the deliberate process of controlling which particular versions of the various applications are permitted to participate in the event federation. There are specific processes for managing each of these types of management. The interrelationship between these processes are shown in Figure 1.

Of note is the process flow ensuring that event driven changes, new capabilities and functionality fixes, are evaluated for insertion into the core baseline version of the simulation, either the proponent baseline version or the BLCSE reference version, or both.

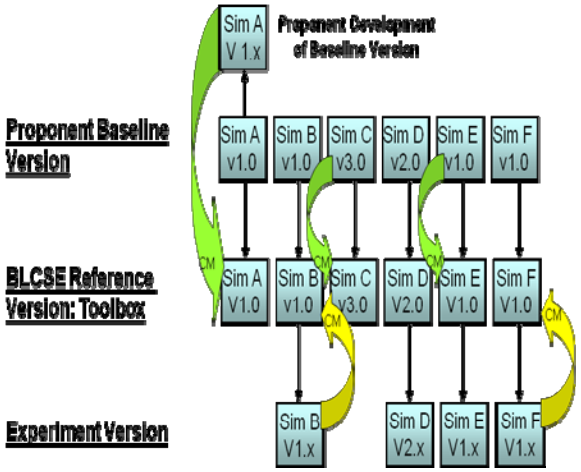

Figure 1: Configuration Control Process

Software Baseline Configuration Management

For OneSAF development, we initially use a PM OneSAF delivered baseline or a version from a

previous event. For other products, like terrain or scenarios, it will depend on the requirements for the event whether we use a previous baseline or start a new baseline.

During the integration process for that event, each product is separately and collectively tested and any issues or deficiencies found are documented and a change request is created and evaluated by the developers and the Technical Configuration Controller(TCC). If the change request is approved, new baselines with the fixes are built and provided to the community for further testing.

Event Configuration Management

While very similar to the software baseline configuration management, event management has several distinct differences. While baseline configuration is generally guided by adding functionality and stability to the product itself, an event version if focused on providing specific functionality needed for a single use case, and on making that functionality operate within a specific environment. Often overall stability and user population considerations are secondary to immediate need.

Event versions are published at a much faster rate than are baseline configurations, which may not publish new versions until a significant single change is completed, or a critical mass of small changes are accumulated. Event versions are generally released on a specified schedule -- normally weekly once later integration is underway -- and may include frequent single-change version releases ("drops") necessary to fix a specific fault. The final significant difference is the end-of-event termination of that version branch, when all changes made during the event are examined for application back to the baseline, and then the remaining changes are archived as dead files.

PROCESS FOR CONTROLLING THE CONFIGURATION OF A FEDERATION.

The key to our event federation configuration control was the development of a specific web portal that functioned as the sole authoritative source of either software or data and settings documents. We have a GOTS (Government Off The Shelf) program (STARSHIP) that will monitor which machines are connected to our RTI and will also query specific files in use on that machine and its information. We use this as an audit tool which allows either the TCC or the event "battle master" to check that the appropriate version of software and Operating System (OS) is loaded and operating.

Roles in BLCSE:

- **Simulation Proponent**: Maintains configuration management of the simulation, including configuration accounting and archive responsibilities. Recommends BLCSE federation version changes to CCB.

- **BLCSE Event Director**: Selects Event Federation from Federation, integrates Event Federation, Recommends Event Federation to CCB, maintains Configuration Management of Event Federation through event end.

- **CCB**: Approves current versions of BLCSE Federation and Federates. Approves Event Federation. Under original charter, only the CCB could approve the construct of the BLCSE federation toolkit or the structure of a BLCSE event federation. Current practice is that the CCB implements DIACAP control over the ARCIC Simlab and federation, while the event director executes sole federation authority.

- **BLCSE Technical Configuration Controller (TCC)**: Maintains configuration accounting and archive functions for current and previous BLCSE Federations and Federates. Archives Event Federations. Maintains Configuration Management of BLCSE Documents and data sets. Determines the configurable items for the BLCSE federation and for each event.

- **Sole CM Authority** Within an event, only one designated individual has the authority to permit a change to any configurable item. While this

would normally be the event director, in practice it is the TCC, with agreement by the event director, who makes a judgment call as to when the impact of making a change is worth the benefit of the change.

Lifecycle versus Event change schedule

As was shown in Figure 1, BLCSE has multiple simulation federates each of which has both baseline and event versions. To reduce the amount of "churn" which would occur, with proponent developers releasing new minor and major change versions regularly, plus the rapid changes which are inherent in a flexible experimentation environment, we developed a distinction between normal lifecycle developments and event driven changes. We tailored the process for implementing these changes based on various lifecycle and event thresholds for changes.

"Trigger" severity

We are using three different trigger thresholds to determine when we will change the approved version of the configurable item. Two of the triggers are fairly pre-planned (version and calendar), while the other is event cause specific, which we call our "patch threshold"

The version threshold can be either external or event driven. Rather commonly, one of the federates will have a change delivered by an external proponent (developer), which require the rest of the federation to make changes, thus driving an overall version change to the overall federation. To prevent an excessive number of changes, we attempt to use configuration control of the federation overall by not allowing the newer version to be inserted into the federation until some other threshold is reached. With four primary simulation federates, failure to use restraint in configuration control would cause federation changes at a monthly or faster rate, which causes unnecessary confusion for the various users.

By using the calendar threshold -- quarterly in our original policy -- we thought to reduce the amount of integration engineering work which would be required. Experience taught us that even by using a quarterly calendar threshold; we were spending too much time moving the federation forward, only to need to make an additional change because of a BLCSE event where a newer version, or specific new functionality, was needed for the event. We therefore changed the calendar trigger from being quarterly to being an event calendar, based on the two or three major distributed events we conducted each year. In this case, the calendar and the version thresholds were effectively combined to be triggered by the calendar start of a new major distributed event, with the latest version of each of the federates being used as a starting point for both the event federation and a configuration control point on the federation baseline.

Our third trigger, the patch threshold, is specific to the event version of the federation, and works at both a short calendar interval (driven by the event) or a severity fix, which is of such a magnitude that the TCC and event lead are in mutual agreement that the fix needs to be done "right now." The patch trigger tends to be a relatively minor change to one or more federates in the event, and is reflected as a minor change in the configuration version number of the federate and federation.

ISSUES IN THE DISTRIBUTED ENVIRONMENT

Our community encompasses different Army Battle Labs, companies, and individuals. Unlike a traditional organization, our developers, test engineers and federates are distributed across the United States and overseas. Event integration spans many weeks leading up to the actual experiment. This has presented challenges to disseminating data, baselines, and documents. Not all of the participants have access to the same corporate or military portals, or have adequate bandwidth or permissions for downloading.

The BLCSE network exists on the DREN (Defense Research and Engineering Network). Because that network is classified, and the BLCSE virtual network is being accredited at a SECRET level, it is preferable to do all development and data storage on

unclassified networks and then transfer the baselines and data to the BLCSE for the testing. This allows for future development and reuse of files on unclassified computers. No single tool allows for version control, file storage, or download capability and bridging the air gap between the unclassified and classified networks. We use many different network tools to accomplish collaboration within our community. Some of them are the Army Knowledge Online (AKO), a SharePoint site maintained on the BLCSE secure network, an ftp server operating on a commercial network, and when there are severe bandwidth limitations or firewall issues and short timelines we will send DVDs via next day delivery to a few sites. For all of these networks, we use designated on-line CM tools and internal control processes to ensure only authorized users access software, and only correct versions are reachable for download.

BLCSE PROCESS

Depending on the event requirements, the initial baseline is chosen either from a prior event, or a new PM OneSAF released version. Once that decision is made, the source software is put into our version control software, and the change notification process is required to make changes to that code.

For event integration we have a weekly change schedule, starting on Monday, and ending on Friday when a new baseline or patch would be published for the community, along with a list of what has changed or any features, updates made. This process includes a Wednesday cutoff of new modification changes, stand-alone software testing, and then distributed software testing as a federate before code lock and version release posting.

During integration testing, if a problem is found, the user identifying the issue is required to fill out a Problem Tracking Report (PTR). This is submitted to the local federate battlemaster, who then notifies the event battlemaster and the TCC. The event battlemaster and the TCC review the PTR for significance, and also for re-creation of the problem, and if further analysis is required. The TCC will either accept or reject the PTR as significant for the event (it may be separately referred to the simulation proponent, but not worked as an issue for that event). If it is accepted, then battlemasters are informed that a change to the baseline is being worked and the programmers review the PTR and decide the best course of action to perform and implement the change. If the PTR is rejected by the TCC then the issue is adjudicated between the local battlemaster and the TCC or the community during the daily hotwashes or other meetings.

The test engineers have until Wednesday to test and document any issues that will require baseline changes for the next week. Anything found after Wednesday cut off time, is shelved for the next week's release unless it is of such a critical nature it will impact the next week's schedule. The programmers test each of their changes locally, but some issues require distributed testing. For those cases, a separate (side) Run-Time Infrastructure (RTI) is set up just to test in a distributed manner, so as to not affect the ongoing integration testing with the Federation. Once all testing is successful, the changes are packaged and processed for release to the community on Friday.

Depending on the nature of the software changes (non-parametric data changes), at the termination of the event, all the PTRs, change files, and code are submitted back to simulation proponent for incorporation in future simulation baseline releases if it passes their CCB process.

Software CM Tools

When we began, our requirements for a tool were fairly simple – we only needed version control. That fit our needs when the programmers would deliver the test baselines via a DVD or .tar file. After a market survey, which considered cost as a determining variable, we found that the Serena's Polytron Version Control System Version Manager (PVCS VM) worked very well within our operational environment and enabled us to manage the changes.

Experience demonstrated that we needed accountability, repeatability, and fault traceability at a much faster speed as we worked in distributed teams across the nation. We did another market survey and selected an upgrade to Serena which was affordable and still met our minimum requirements. It was not our first choice on a pure technical basis, there were better tools available, but we were constrained by explicit budget restrictions.

BLCSE is currently transitioning from Serena's PVCS VM product to its full featured Software Configuration Management (SCM) product, Dimensions CM. At present we are still using the PVCS VM product, and will retain that tool as a basic version management tool for users. Dimensions CM will be our primary production configuration management tool for development engineers. While having a single, all-purpose CM tool which supports both production configuration management plus user version release (configuration control) and detailed archive functionality is desirable; using multiple management tools, by one vendor or by multiple, may be the most cost effective solution. The authors are not advocating community acceptance of Serena products, but are describing the attributes which determined it's suitability in our environment. CM teams need to examine the tool requirements of their particular environment in a market survey and determine which tool best suits their needs.

In considering the requirement for configuration control tools, our experience has shown two critical technical considerations in selecting the right tool, separate from the best value consideration. The first is the degree to which multiple users will be accessing the controlled baseline and making changes to that baseline. The configuration authority has to balance between giving multiple users simultaneous access to the baseline, although not the same block of code, and ensuring that changed code blocks are checked back in with appropriate testing and notifications. The authority to control actual baseline changes (new baseline version) while allowing innovative software development to a "test" baseline is also critical.

The second technical consideration is the ability to document not only the actual code change but the rationale and backup materials for later re-integration of those changes into a proponent baseline, so that the work and knowledge of making event changes can be appropriately handed off to the proponent owner of the simulation. This functionality needs to be fairly simple and quick for the code developers, who are under time pressure to make event changes very quickly; and yet rigorous enough to meet the demands of deliberate configuration control boards managing a proponent baseline.

Initially we invited the BLCSE community to use our version control software if they needed that capability. Most sites have developed their own internal processes and declined to use it. Since most of our CM requirements for the events and community have been met by the use of the SharePoint portals, AKO, we've not expanded the role of collaborative development beyond our own internal group. Our intent is that as the community development grows, we'll expand our use of our CM tools to facilitate that effort.

We found that Dimensions CM provided the ability to audit, manage baselines, capture requirements and develop in parallel; and merge changes in one tool, rather than manually hand jamming data and hoping it all ties together correctly for reporting. We can also maintain different event baselines at the same time and provide multiple releases for future events. Dimensions CM also has plenty of different software integrated development environments (IDEs) for programmers so they don't need to change the way they program to learn a tool. The design of a global database repository has increased the transfer speed across the WAN and the use of metadata and library caching helps programmers keep their code synched to the repository. For software version releases, it also has automated capability for pushing baselines, and email notices on changes to designated users.

Dimensions also meets the requirements for information assurance security. It can authenticate users with CAC cards, a requirement which we are transitioning to as part of our DIACAP accreditation. Its ability to meet compliance requirements and enforce process management and still provide complete traceability helps us also meet the stringent security requirements we now face with DIACAP.

DIACAP CCB/CM requirements

As part of Department of Defense regulations, DIACAP has significantly impacted the way we perform configuration management and configuration control. At the very basic level, DIACAP mandates that we have a configuration control board overseeing what is approved to be part of the BLCSE simulation and tools federation. But beyond this, the need for accreditation, and for certification of persons having change authority of the software and hardware/Operating System (OS) platforms, has had dramatic impact.

Operating System Impacts

Under DIACAP, the operating systems on our hardware are regularly checked with DoD provided information assurance tools. Experience has shown that, given the inherent, albeit necessary, paranoia of the Information Assurance (IA) community, each new release of the scan tools routinely detect a large number of new vulnerabilities, requiring the operating system to be "fixed." The trouble is that quite often, these OS changes end up affecting the simulation software, which then also needs to be changed via a patch, which must be approved either by the TCC or by the CCB, based on the level of that change. Since these IA scans come out much more frequently that do routine or scheduled updates to the simulation baselines, the CM authority must strike a difficult balance between meeting all of the IA forced OS changes, and keeping the simulation software working properly on the operating systems.

Change accounting

The other side of DIACAP is that all software is required to be under full change accounting, as well as only be changed by persons having authority to do so. Change accounting, also called production control, is an inherent part of good configuration management. Just as the Version Description Document describes what changes are made in each new version, the change accounting process identifies who made the various changes, and also who approved and authorized the changes. The requirements for change accounting were part of the rationale for moving from our basic Version Manager CM tool into the more advanced Dimensions tool suite.

SUMMARY

Interoperable simulation events require all participants to share a common picture of the environment. Configuration management and configuration control are the processes by which we control the rate of change in our configurable items and ensure all participants are using the same versions of software and data.

Configuration management and configuration control can be either very flexible or extremely rigid and prescribed. The needs of the operating environment should drive the structure of the processed. The configuration management authority needs to find the balance between flexibility and control based on the needs of his particular environment. The number of configurable items within a simulation event must also be found between making all documents configurable items, and leaving just the software suite as a single configurable.

The configuration management authority needs an appropriate tool set to assist in the physical process of controlling and managing software baselines, versions and other configurable items. The toolset should include both basic production control/version management software as well as archive and version release/publication tools or functionality. There is no one "best tool" for all needs, but instead, based on the operating environment tools are selected that are appropriate and meet requirements. And the tools

themselves may need to change as the requirement or operating environment changes.

The greatest challenge the configuration management authority will face is to ensure that the processes and tools serve the needs of the users, which is to enable the simulation event and the software developers, rather than become a restrictive straitjacket that prevents timely software and document updates to meet event objectives.

ACKNOWLEDGEMENTS

The authors would like to thank Robert (Ben) Abel, of Expeditionary Technologies, for his editorial contributions to this paper.

REFERENCES

Haas, Anne Mette Jonassen (2003). *Configuration Management Principles and Practice.* Boston: Addison-Wesley,