Proceedings of the 2010 Winter Simulation Conference
B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan, eds.

SYSTEMS ENGINEERING FOR DISTRIBUTED LIVE, VIRTUAL, AND CONSTRUCTIVE (LVC) SIMULATION

Scott Gallant

Effective Applications Corporation
318 Osprey Lakes Circle
Chuluota, FL 32766, USA

Chris Gaughan

U.S. Army Research, Dev. & Eng. Command
Simulation & Training Technology Center (STTC)
12423 Research Parkway, Orlando, FL 32826
Orlando, FL 20052, USA

ABSTRACT

Designing a distributed simulation environment across multiple domains that typically have disparate middleware transport protocols, data exchange formats and applications increases the difficulty of capturing and linking system design decisions to the resultant implementation. Systems engineering efforts for distributed simulation environments are typically based on the middleware transport used, the applications available and the constraints placed on the technical team including network, computer and personnel limitations.

To facilitate community re-use, systems engineering should focus on integrated operational function decomposition. This links data elements produced within the simulation to the functional capabilities required by the user. The system design should be captured at a functional level and subsequently linked to the technical design. Doing this within a data-driven systems engineering infrastructure allows generative programming techniques to assist accurate, flexible and rapid architecture development. This paper describes the MATREX program systems engineering process, infrastructure and path forward.

1 INTRODUCTION

Simulationists who require the use of distributed simulation typically do not have a long life cycle for an experiment, analysis initiative or simulation-based event. To reduce cost, they need to use a well-established simulation architecture and robust models that are easy to integrate with other distributed simulations. This short lead time for system design, development, integration and execution forces the system definition and design to happen very quickly.

These simulation users rely on standards and influence to simulation developers to get the systems to communicate using the same syntax. This often works to instantiate a System of Systems (SoS) architecture (Jamshidi 2008) and get models to share information. A SoS environment is an assembly of applications that together provide more capability than the sum of their individual capabilities. Within the M&S community, the applications assembled are focused on representing a specific warfare function based on data and models from an organization considered to be the center of excellence for that function. The SoS architecture provides many benefits above a single instance of Semi-Automated Forces (SAF) modeling including performance, model management and information transparency for analysis.

However, the biggest problem in these cases is that the models do not work together semantically for the accomplishment of the high level functions that the users require. In other words, applications may not be communicating based on a consistent understanding of the context and connotation of the information being shared. We have developed a tool that ensures semantic interoperability traced back to functional requirements. We have learned many lessons in our work and have some ideas for the future of Systems Engineering SoS architectures.

Our intended path forward is to increase the use of generative programming techniques, or automatically generating executable computer programming artifacts from a higher level source, in order to quickly deploy a SoS architecture for military analysis. The flexibility required to implement our goal requires systems architecture qualities and objectives such as encapsulation of functionality into appropriately sized portions to be able to manipulate and construct larger capabilities as needed with as little engineering effort as possible. We'd like to provide an architecture that is fully compliant with U.S. Army-grade verification and validation guidance (DAU 2008) and robust enough for decision-oriented analysis while maintaining flexibility and quickness in order to save the Army tremendous amounts of time and effort (Page et al. 2001) when constructing distributed Modeling & Simulation (M&S) environments for various uses.

1.1 MATREX Overview

The Modeling Architecture for Technology Research and EXperimentation (MATREX) provides a unifying distributed Modeling and Simulation (M&S) architecture and supporting tools and resources that ease the integration and use of multi-resolution live, virtual, and constructive (LVC) applications. It enables full spectrum analysis of system designs and operational concepts while reducing risk and acquisition timelines (Tufarolo et al. 2004). To provide this capability across the Army and the other Services, the Assistant Secretary of the Army for Acquisition, Logistics, and Technology (ASA(ALT)) and the Research, Development & Engineering Command (RDECOM) have funded development of the MATREX program. The MATREX environment, including the toolset and resources, is available as Government-Furnished Software (GFS) with some support and training available.

An important aspect of MATREX is its capability to leverage many M&S resources developed at RDECOM laboratories, centers and activities as well as the larger Army and other Services. The world-class RDECOM team of M&S experts has utilized MATREX to identify issues, exercise requirements, align development efforts and conduct experiments that involve SoS technologies beyond the expertise of any one part of RDECOM.

Additionally, the MATREX environment is used for a variety of purposes. These numerous and often generic uses of MATREX offer a difficult systems engineering challenge in linking system requirements to detailed system design and technical dependencies. To facilitate customization for a given user's goals, the MATREX systems design team has developed a highly flexible and configurable system design that will allow it to meet a wide breadth of user requirements.

1.2 System Design Description Infrastructure Overview

The MATREX suite of models, tools and architecture allow for many different possible configurations of the system to achieve the user's functional requirements. The philosophy is to work with the users to develop a System Design Description (SDD) that meets their exercise requirements, data decomposition requirements, system architecture guidelines, scenario, configuration choices and model selection.

The MATREX program has developed its general SDD to capture the system design at a functional level and subsequently link the functional design to the technical design. This allows the functional requirements to be linked to system design and allocated to specific models as shown in Figure 1.

The low level requirements, object model and test cases can then be auto-generated based on model allocation to functions. The SDD is data-driven, easing information maintenance duties by linking the system engineering products and simplifying the editing of the system design. The SDD also allows for auto-generating low level specifications, certain Department of Defense Architecture Framework (DoDAF) (DoD CIO 2009) views and test cases. The SDD includes the mapping between the data to be collected during the exercise, the initial exercise goals and the explanation of what the data means.

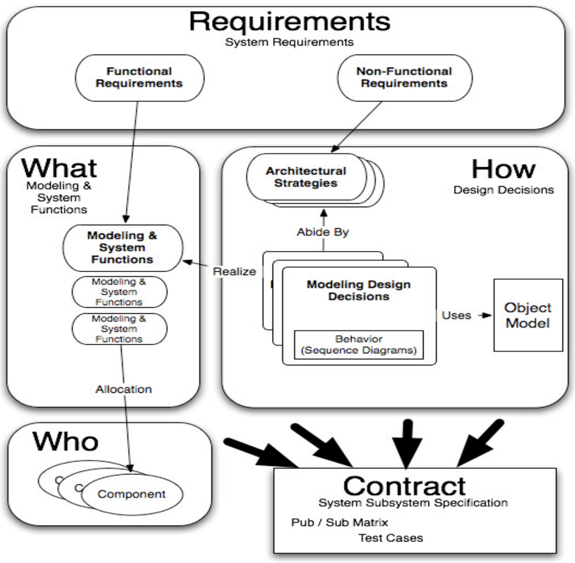
Requirements
System Requirements
Functional Requirements
Non-Functional Requirements
What
Modeling & System Functions
Modeling & System Functions
Modeling & System Functions
Modeling & System Functions
Allocation
How
Design Decisions
Architectural Strategies
Abide By
Realize
Modeling Design Decisions
Behavior (Sequence Diagrams)
Uses
Object Model
Who
Component
Contract
System Subsystem Specification
Pub / Sub Matrix
Test Cases

Figure 1: Database driven systems engineering infrastructure to link SoS information for an accurate and automated system.

1.3 MATREX Tools Overview

Two important tools that we use to export executable system tests and ultimately more architectural or implementation functionality are the MATREX ProtoCore and MATREX Advanced Testing Capability (ATC).

1.3.1 MATREX ProtoCore

The MATREX ProtoCore offers an Application Programmer's Interface (API) that abstracts away the middleware protocols in order to allow the developers of models to focus on model behavior vice rewriting code to align with specific distributed simulation protocols. This API (Snively et al. 2006), reduces reliance on middleware services, data management techniques and architectural design patterns. In turn, it allows a simulation event to be middleware agnostic as the automated tool suite helps set up and configure models based on the SDD. Because ProtoCore offers a forward path for legacy object models and component models, future users can leverage existing software investments and minimize expenditure in development cycle resources by using ProtoCore to port their models to current distributed simulation standards, such as High Level Architecture (HLA) 1.3 Next Generation (NG), HLA 1516, Test and Training Enabling Architecture (TENA) and One Semi-Automated Forces (OneSAF) Simulation Object Runtime Database (SORD).

1.3.2 MATREX Advanced Testing Capability

The MATREX Advanced Testing Capability (ATC) is a testing tool that generates test applications based on the design and over-the-wire communication distributed simulation requirements (McCray et al. 2008). The ATC allows for testing over the middleware layer based on a predefined sequence diagram including validation values. The ATC's primary purpose for MATREX integration testing is its ability to perform meaningful and repeatable "black box" level testing on any component being integrated into the MATREX environment (see Section 2.2.3). This allows a simulation to test interactions with other simulations without needing to run them in turn, reducing integration costs and test complexity. ATC also performs the function of documenting specific test cases in order to provide reproducibility. The goal for the near future is to export sequence diagrams from the SDD into a standardized format for ingestion into the

ATC tool. This will allow for testing components directly from the design and technical contracts linked to system requirements further reducing integration costs.

2 AUTOMATIC GENERATION OF ENGINEERING ARTIFACTS

2.1 Current Community Issues

There are many critical qualities that managers of a simulation environment must achieve: traceability from requirements to implementation and the resultant data collected, alignment of data semantics across applications, ease of maintenance and change propagation throughout the architecture. Aligning data semantics is referring to ensuring applications are communicating based on a consistent understanding of the context and connotation of the information being shared.

When integrating existing applications that are chosen because they share a common syntax, or even for political reasons, e.g. someone with the authority orders the use of a model, the integration of applications must be backward engineered to the functionality required. Systems are often chosen because of the object model and middleware protocol that they are compatible with. However, compatibility is more than the ability to communicate without compilation errors or crashing. The applications' capability must provide necessary portions of a high level capability and they must provide that functionality in semantic harmony with the other applications within the architecture.

We have been involved in dozens of events and exercises and in every single one, we have witnessed changes to the implementation throughout the integration and preparation. Most of the time, heavy change is still required up to only a few days or hours before the start of execution. Engineers often pull off technical miracles at the last second including working through the night or using one-time fixes that they know are not good long term solutions. Sometimes those changes work out, but frequently they are the cause of reduced availability, reliability and effective modeling.

2.2 Benefits of Automatic Generation of Engineering Artifacts

We have made great strides in implementing some of the core building blocks for generative programming techniques to the distributed M&S domain. It has become clear to us that there are exponential returns on investment when supporting the design and implementation of distributed M&S environments. We are planning on achieving some of these strategies and we are always exploring the successes of others to improve on our solution for broader application within the military M&S community.

Capturing the Systems Engineering data within a database-driven infrastructure has allowed us to have full traceability from the top-level functional requirements through the design and implementation choices through to the detailed technical engineering artifacts used by all phases of the exercise implementation. The engineering artifacts include detailed technical requirements, systems engineering views for design discussions and even executable test cases. The next step will be to generate artifacts that can be used by the simulation and management applications as shown in Figure 2.

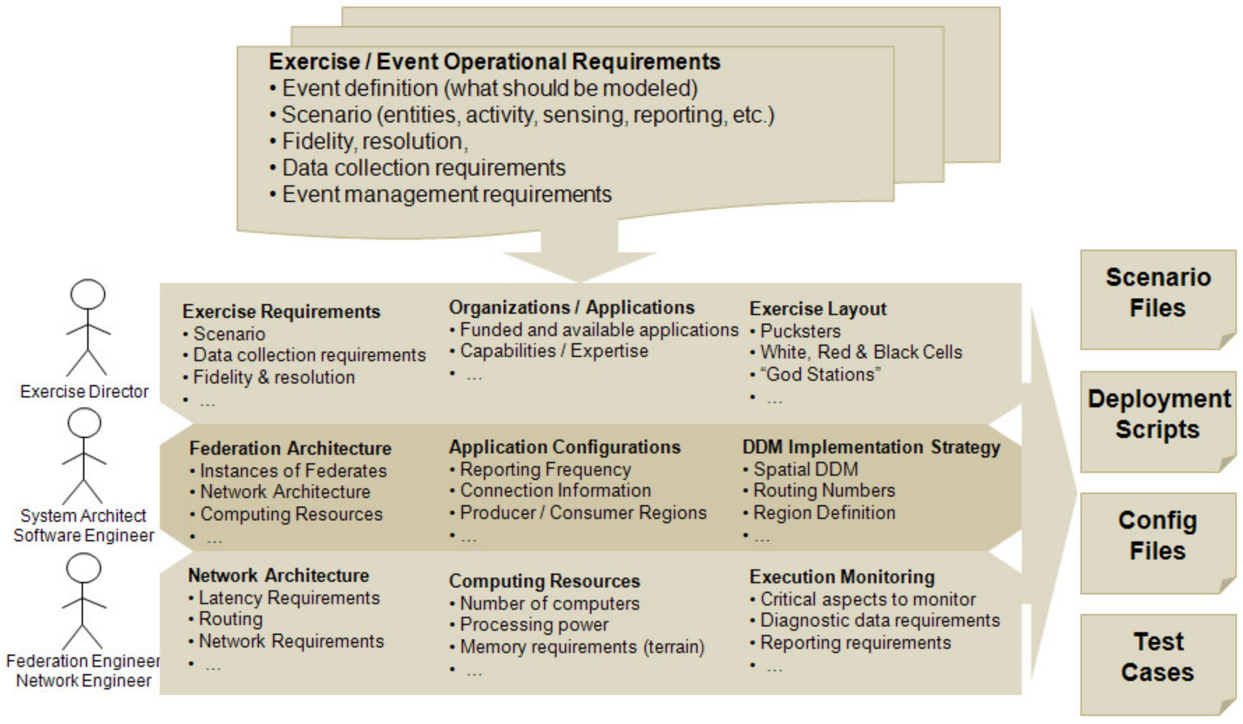

Figure 2: An approach for automating much of the event life cycle.

2.2.1 Top-Down Systems Engineering Benefits

Capturing the modeling requirements through a top-down decomposition ensures that the engineers understand the functions and information exchanges that are required to accomplish the high level modeling functions. Whether there are applications that can meet those needs or not, the engineering staff understands where there are weaknesses or workarounds necessary.

Applications and interface messages can later be allocated to the functional decomposition. The applications' ability to meet the functional requirements ensures traceability from the implementation back to the functional needs which can then be tied back to the purpose of the system as a whole.

2.2.2 Code Generation for the Object Model

The MATREX manages an HLA Federation Object Model (FOM) that many organizations throughout the military modeling and simulation (M&S) community use, including RDECOM, the Training and Doctrine Command (TRADOC) and the Army Test & Evaluation Command (ATEC). That FOM evolves and changes every six months through a formal review and release process.

The MATREX ProtoCore includes a code generation capability to turn the FOM into a set of programming classes that can be used by application developers. By centralizing the generation of classes based on the evolving FOM definition, there is less reliance on application developers to all make the same changes accurately. This also saves time and eases developer participation within the environment. If we had to rely on every developer to make the right changes and make them quickly, we would have a much larger management burden than just providing the new classes out to the developers in a single distribution.

2.2.3 Advanced System Black-Box Testing

The ATC allows the user to build tests within its graphical user interface (GUI) to test each system interface. It stores these tests in an Extensible Markup Language (XML) file. The XML file format is easily replicated by the MATREX SDD so that our ATC test cases are explicitly generated from systems engineering decisions and design captured in our systems engineering database.

This alleviates a great deal of time for our integration and testing staff by avoiding the need to manually change hundreds of tests and test processes due to a few small design changes. The code generation integration of ProtoCore with the ATC also means that object model changes are easy to adapt to over the evolving versions and instantiations of the MATREX architecture.

The ATC can test each system individually according to the design captured within the systems engineering infrastructure. These independent tests eliminate the complexity of a SoS architecture and can isolate interface details in an easy to execute testing environment. Since the tests are automatically generated from the design phase, test cases can be distributed to the developers the same day that the engineering decisions are being made. Development teams can code to the provided tests rather than spend their own time developing independent and possibly erroneous tests.

Test cases can be developed that test a subset of the systems to be integrated to increase the scope of the testing while maintaining an appropriate level of isolation from the complex SoS environment. Testing threads can further diagnose problems when integrating disparate applications built by numerous development teams.

2.2.4 Architectural Design Agreements

In SoS environments like the MATREX, we rely on a Federation Agreements Document (FAD) to explain architectural interoperability agreements to all the developers that need to integrate their applications. That document captures agreements on the use of coordinate systems, dead-reckoning and the heartbeat timing and distance thresholds for objects .

The MATREX ProtoCore software library is already used by many applications as their tool for interoperability with the simulation middleware protocol so adding architectural compliance was the next obvious step. We have added capabilities such as coordinate conversions and dead-reckoning (Aronson 1997) to the MATREX ProtoCore. The additions can be made more flexible if they were driven by the systems engineering infrastructure. If architectural design decisions could be automatically driven by the systems engineers it would further decrease the time and chance for discrepancies in application adjustments as the technical characteristics of the system are changed.

The design for dead-reckoning was kept dynamic so an operator could change the dead-reckoning distance and timing thresholds during run-time. This allows the operator to control the execution performance and accuracy from a central point. This can be used to recover the system from technical issues of slow performance, to accelerate the scenario without flooding the network or to change the dead-reckoning attributes of forces based on spatial considerations such as prioritizing updates for entities within an area of interest for the analysis of the scenario.

2.2.5 System Configuration

Configuration of the participating systems within a SoS architecture are both application-specific as well as based on the scaling strategy and scenario that will be executed. By capturing and linking application configuration requirements to scenarios and functional requirements for the architecture, we could map configuration options to the appropriate use and automatically export the proper configuration depending on the scenario and functions the systems engineers require for any given instantiation of the architecture.

The M&S infrastructure such as the middleware and its configuration is also based on the scale and architecture of the implementation required. As engineers are making design decisions within the systems engineering infrastructure, the middleware configuration is predictable and can also be automatically exported from the systems engineering tool.

2.2.6 Integrated Scenario Development and Initialization

The MATREX SDD is currently limited to the functions required rather than the scenario in which those functions will be used. As we add the scenario information within the SDD, we will be able to further predict the appropriate system configuration to execute the scenario. We will also be able to remotely initialize the SoS environment from a central point.

Our design pattern for simulation initialization is to provide the scenario details such as initial platform attributes and force laydown over the simulation middleware at the beginning of the exercise. The structure of the information and the design paradigm are already in place to expand the SDD to incorporate scenario information and export the necessary engineering artifacts to configure simulation management tools, such as simulation initialization, data collection and execution monitoring. We already use community standards for the storage of the scenario information so integration with popular community tools will be straightforward.

A subtle benefit for incorporating the scenario information within the systems engineering tool is that in many cases the functional design depends on how the scenario will be executed. Similar to how military operations depend on the mission, the execution of the mission with M&S applications also depend on many parameters, many of which are based on the scenario.

2.2.7 Integrated Application Deployment

We can already remotely deploy applications to lab machines and we can even remotely launch applications based on a set of well defined configuration files. We will be expanding this capability with a look at virtualization technology to improve the automation and flexibility based on the systems engineering design and configuration choices.

As the automation increases in this area, the ability to execute analysis events without large efforts from engineers improves. We can imagine a future that includes an analyst using the systems engineering tool to design, deploy, configure and manage the SoS architecture based on accredited models. The execution run could occur on a representative set of lab machines that can be setup on the fly to accommodate the captured execution configuration. These machines could be at various geographic locations with results be compiled and sent to the analyst automatically when complete.

2.2.8 Active Design-Based System Monitoring

As we capture more information about the execution environment, more information will be available to recognize when the system is performing adequately and when the system is beginning to fail. Active monitoring and system management can help engineers recognize issues early so they can fix them without a lot of wasted execution time and cycles.

Monitoring information exchanges at run-time ensures that the implementation does not deviate from the design. A part of the monitoring includes monitoring performance via response time of applications, queue lengths of applications sending and receiving information and machine diagnostics such memory footprints and processor loads.

3 APPLICATION OF GENERATIVE PROGRAMMING

Generative programming is the act of automatically generating executable computer programming artifacts from a higher level source. The benefits include an ability to represent the execution concepts and engineering decisions within an easier to manage format rather than having to manage code which can become laborious and tedious. Generative programming, also known as automatic programming is not new to the software engineering discipline (Czarnecki et al. 2000). It has matured over many years enough that lessons learned can be confidently applied to distributed M&S. We believe that the foundation that we have created on the MATREX program for automatically generating executable test cases and engineering details could prove to be the first step in the further application of generative programming techniques described within this section.

3.1 Approaches on Domain Analysis for a Domain Specific Language

Warfare changes as rapidly as fast, or even faster than technology. M&S architectures are often built to meet only a subset of warfare and they are usually built to study different issues within an area of warfare. An organization's analytical focus drives functional requirements for the simulation SoS. The resolution and fidelity of their focus differs depending on their intended outcomes and their areas of expertise.

Many projects have attempted to build domain models (Evans 2003) of warfare with varying levels of success. Domain modeling is the act of modeling a particular capability specific to a discrete domain. However, warfare elements cannot be discrete when analyzing how the deployed systems are going to interact with other deployed systems, an unpredictable environment and an unpredictable enemy. The use of those domain models to any simulation environment is always riddled with difficulties. The subject matter experts (SMEs) for their specific area of warfare modeling almost always disagree with either the accuracy or the level of resolution of the domain model.

Warfare changes too rapidly for a single referential domain model to exist. A major strength of United States Armed Forces is their ability to adjust to our enemies. Combating diverse enemies in different parts of the world requires unique tactics, techniques and procedures (TTPs) for each theater. Even within a single branch of the Armed Forces, the way they accomplish their mission can differ between theater operations. Furthermore, tactics within the same theater have changed over short periods of time. For example, the networked fires procedures of US troops in Iraq have been changed as the threat has changed in composition and reaction to US forces.

3.2 Areas for Application of Aspect-Oriented Techniques

Aspect-Oriented Programming (Jacobson et al. 2005) is a strategy for increasing modularity of software by separating cross-cutting patterns from core business logic. The encapsulation of broad and repeated application concerns provides flexibility and easy adjustments to implementation details that are necessary but independent of the core business logic.

The majority of military models have been built for stand-alone use by their project offices. The integration of models into a SoS architecture is typically an afterthought. The base business logic of the applications are often integrated with distributed simulation protocols such as Distributed Interactive Simulation (DIS), HLA and TENA, and their respective object models after the models have already proven useful as a stand-alone tool for analysis. This add-on functionality is done to meet the protocol specification and often based on a specific environment's architectural guidelines for interoperability elements like dead-reckoning and heart-beating.

The add-on nature of an interoperability library lends itself to aspect-oriented programming since the core business logic (the actual modeling rather than the software written to provide the infrastructure for the model) is already separated from the simulation middleware details. Further encapsulating specific portions of the simulation middleware interoperability logic can lead to better flexibility in making adjustments as necessary when migrating an application into additional SoS environments or as an architecture evolves over time.

Furthermore, if the aspects for architectural interoperability were driven by the Systems Engineering toolset, then changes could be made very quickly and adjusted based on testing results or any late notice technical requirement alterations.

3.3 Code Generation from Pseudocode in a SoS Architecture

In order to allow for easier modification and traceability from the logic to both functional requirements and design decisions the functional logic can be captured in Pseudocode. However, capturing pseudocode in a format that can generate executable code (Roy, 2006) is non-trivial for domain agnostic applications. The limitless possibilities of the implementation of software require pseudocode definitions and code generation tools to account for a great deal of permutations. This results in lengthy and detailed pseudocode that could be equally represented with fourth generation programming languages. Fourth generation programming languages like the Scala programming language (Wampler et al. 2009), provide tighter control of syntax guidance and problem notification during compilation.

The narrow breadth of the military M&S domain and implementation choices provide us the benefit of limiting possible permutations of the generated software. We already have software libraries to assist in the middleware protocol compatibility that we could configure and initialize from the same source as the pseudocode translation to software.

The previously mentioned capabilities to export configuration information for the middleware connection library provides the basis for the generated code. The ProtoCore software library already provides code generated classes that represent the object model and a set of static API methods that can be built into the Pseudocode. The ATC tool can already generate a system shell that successfully joins the middleware and sends and/or receives information. The foundation for creating a working application within our SoS architecture is already in place.

Since we already have a mechanism to automatically generate the computing infrastructure to connect to the architecture middleware, we have separated the functional logic from the foundation of the software application. This practice provides the most benefit when algorithms can be separated from the data used within the algorithm. We could capture the functional logic and algorithms of models within Pseudocode and retain the data files separately. The verification and validation of the application needs to occur within the SoS environment and for the scenario that will be used. Verification and validation becomes easier if the application's structure, functional logic and data are customized for the SoS environment that it will be run in.

The subsequent work remaining involves capturing function execution within Pseudocode that is based on the interfaces that our systems engineering tool manages. For our environment, models require extensive and stringent accreditation based on subject matter expertise and validated algorithms. Generation of mature models will be difficult and a goal for the long term, but in the meantime generation of surrogate applications to replace unavailable applications or fill gaps in the SoS architecture will be beneficial.

4 PERCEIVED LIMITATIONS AND RECOMMENDATIONS FOR THE FUTURE

The value of generative programming techniques has been tremendous for integrating disparate systems that are separately funded and managed. However, there is a limit to the semantic integration of models when we can only translate and manipulate the results of the models through their interfaces. The models themselves just do not align in some cases.

Many projects have attempted to provide translations between object models to facilitate interoperability between models that work on different object models. The problem with that approach is that the models do not provide a quality mapping between their internal structures and any object model because the mapping is syntactical matching of best available options. Interoperability goes well beyond the ability to communicate with common data structures (Tolk et al. 2003) but extends through and beyond semantic synchronization.

A framework for structuring the integration of atomically encapsulated military functions could provide the ability for model developers to implement their software applications within a well-structured suite of software building blocks that could be rearranged as necessary to meet high level requirements. This framework would need to provide an ability to connect functions via information transportation. The systems engineering of such a framework would need to hierarchically organize and connect functions to account for heavily related processing dependencies and reuse of often used information. For example most military functions rely on ground truth of the entities to be effective. The modeling of entities and the resultant updates referred to as ground truth need to be provided to multiple recipients where each recipient would likely use the information differently. Each of the intended uses of that information need to be captured within their respective function interfaces. The requirement of publishing ground truth is the union of all the user interface requirements. The framework needs to be focused on information use so many of the available data elements would not need to be published when no subsequent military function would need to use the information to execute their respective models.

ACKNOWLEDGMENTS

This paper is based on work executed over the last four years for which we'd like to acknowledge some of the personnel most involved in its development and maturation. Those personnel are: Sid Antommarchi, Tracey Beauchat, Jon Clegg, Jim Gallogly, David Itkin, Rich Leslie, Richard Mangieri, Lee Mangold, Tom McAfee, Paul McCray, Rachel Offutt, Keith Snively, Jeff Swauger, Jeff Truong, and John Vintilescu. We'd like to also acknowledge our management personnel that helped provide vision, resources and encouragement: Chris Bailey, Tom Hurt, Tom Mathis, Joe McDonnell, Chris Metevier, Dave Poole, and Gary Smith. Finally, we'd like to acknowledge ASA(ALT) and RDECOM for providing the funding necessary to continue these critical efforts.

REFERENCES

Acquisition Community Connection at Defense Acquisition University (DAU). 2008. *MIL-STD-3022 DoD Standard Practice Documentation of V&V for Models and Simulations*. Available via https://acc.dau.mil/CommunityBrowser.aspx?id=205916

Aronson, J. 1997. *Dead Reckoning: Latency Hiding for Networked Games*. Available via http://www.gamasutra.com/view/feature/3230/dead_reckoning_latency_hiding_for_.php

Czarnecki K. and U. Eisenecker. 2000. *Generative Programming: Methods, Tools, and Applications.* 1st ed. Addison-Wesley Professional.

Department of Defense (DoD) Chief Information Officer. 2009. DoD Architecture Framework 2.0. Available via http://cio-nii.defense.gov/docs/DoDAF%20V2%20-%20Volume%201.pdf.

Jacobson I. and P. Ng. 2005. *Aspect-Oriented Software Development with Use Cases.* 1st ed. Addison-Wesley Professional.

Jamshidi, M. 2008. *System of Systems Engineering*. 1st ed. Wiley.

McCray, P. and K. Snively. 2008. Functional Component Testing for Distributed Simulations. *Simulation Interoperability Workshop Spring Conference, April 2008*

Page, E. and D. Lunceford. 2001. *Architectural principles for the U.S. Army's simulation and modeling for acquisition, requirements and training (SMART) initiative.* 2001 Winter Simulation Conference (WSC'01) - Volume 2, 2001 pp.767-770.

Roy. G. 2006. *Designing and Explaining Programs With a Literate Pseudocode*. Journal on Educational Resources in Computing (JERIC) by Association for Computing Machinery (ACM). Volume 6, Issue 1 (March 2006). ACM, New York, NY USA

Snively, K. and P. Grim. 2006. ProtoCore: A Transport Independent Solution for Simulation Interoperability. *Simulation Interoperability Workshop, September 2006*

Tolk, A. and J. Muguira. 2003. The Levels of Conceptual Interoperability Model. *Simulation Interoperability Workshop Fall Conference, September 2003*

Tufarolo, J., R. Leslie and D. Lewis. 2004. *Distributed Integration for the V0.5 MATREX.* Simulation Interoperability Workshop, Spring 2004, 04S-SIW-131.

Wampler, D. and A. Payne. 2009. *Programming Scala*. 1st ed. O'Reilly Media

AUTHOR BIOGRAPHIES

SCOTT GALLANT is a Systems Architect with Effective Applications Corporation. He has over fifteen years experience in distributed computing including Army modeling and simulation. He has led technical teams on the MATREX program for distributed software and federation design, development and execution management in support of technical assessments, data analysis and experimentation. He is leading the MATREX Systems Engineering team for the implementation of the described product infrastructure. Scott Gallant also currently serves as the System Architect for the MATREX program. He earned his Bachelor of Science in Computer Science from George Mason University. His email address is <Scott@EffectiveApplications.com>.

CHRIS GAUGHAN is the Deputy Technology Program Manager of the MATREX program at the Simulation & Training Technology Center within the US Army RDECOM. From 2004-2009 he worked at the Edgewood Chemical Biological Center where he served as the Configuration Manager of the Chemical-Biological-Radiological-Nuclear Simulation Suite. During his tenure at ECBC, he has been the principal investigator for numerous Joint Science and Technology Office projects focused on CBRN M&S. He has provided M&S analytical support to the Joint Program Executive Office for Chem-Bio Defense and to the TRADOC Maneuver Support Battle Lab. He received his Master of Science and Bachelor of Science in Electrical Engineering from Drexel University in Philadelphia, PA. His email address is <Chris.Gaughan@us.army.mil>.