Proceedings of the 2011 Winter Simulation Conference
S. Jain, R.R. Creasey, J. Himmelspach, K.P. White, and M. Fu, eds.

DIFFICULTIES WITH TRUE INTEROPERABILITY IN MODELING & SIMULATION

Scott Gallant

Effective Applications Corporation
Orlando, Florida

Chris Gaughan

U.S. Army Research Laboratory (ARL)
Simulation & Training Technology Center (STTC)
Orlando, Florida

ABSTRACT

Interoperability among distributed models and simulations is complex, tedious and difficult to evaluate. Integrating models that were developed for various purposes with disparate technologies and managed by independent organizations is often the goal. This goal is underestimated due to misleading facts of commonalities between those applications. Common compliance with middleware architectures, modeling goals and even object models gives a false impression of complete interoperability. There are numerous considerations when developing a distributed simulation environment. The event's objectives drive the necessary simulation functions, but how those simulation functions interact needs to be meticulously designed for true interoperability. The semantics of the information transmitted, the behavior necessary across multiple applications, fidelity and resolution synchronization are only a subset of the systems engineering necessary for a coherent System of Systems. This paper covers interoperability complexities and proposes criteria to consider when developing, integrating and executing a distributed modeling and simulation architecture.

1 BACKGROUND

Technical integration frameworks evolve quickly. The technical leaders in the distributed modeling and simulation (M&S) industry cannot agree on the best technical path forward at any given time. Each program that develops a model or simulation has a specific purpose, set of requirements and limited funding. These programs cannot afford to coordinate with the entire industry to provide the most "interoperable" product. Even if they did meet with everyone to gather interoperability requirements, the task list would be too large to execute with limited funding. In some cases, the requirements would be mutually exclusive in the eventual implementation. The program offices should budget for and plan for coordination across domain projects within a limited scope to improve interoperability with those applications that are most appropriate, but the utopian goal of cross-Service and cross-domain interoperability cannot be solved by funding alone.

We work on an Army project that has integrated dozens of models into a single System of Systems architecture. These models have a wide range of purposes, fidelity, resolution, managing organizations and technologies. The topics covered herein are inspired by lessons learned on this project, as well as our collaboration with many other efforts in the M&S community.

2 INTEROPERABILITY

Interoperability, described most simply, is the ability for multiple systems to work together for a larger goal than any one system could provide on its own. Some simple examples of interoperability by two disparate components include USB drives made by a multitude of manufacturers working with a computer, cell phones made by different companies using different communication protocols being able to allow da-

ta transmission by users, and multiple web browsers being able to interpret and show web pages served up by disparate web servers. These types of simple examples are made possible by the use of standards that can be conformed to by software and hardware providers.

Standards in M&S cover multiple layers of technical abstraction. There are middleware specifications, such as the High Level Architecture (HLA) (IEEE Xplore Digital Library 2010), that dictate how bytes are transmitted between applications. There are standards for simulation initialization, such as the Military Scenario Definition Language (MSDL) (SISO PDG MSDL 2011), which standardizes military scenario representation. There are even standards for how to efficiently exchange manufacturing lifecycle data, such as the Simulation Interoperability Standards Organization (SISO) (SISO-STD-008-2010) Core Manufacturing Simulation Data (CMSD) (SISO PDG CMSD 2011), and how to structure an object model (data exchange format and definition for distributed applications to share information), such as the Object Model Definition (OMD) file format in HLA.

Standards enable a critical key to interoperability allowing simulation services to cooperate together. However, standards alone do not allow for interoperability due to the rapidly changing and drastically expanding breadth of capabilities used in simulation-based events. Standards take time to develop and approve across the M&S community while events have shorter schedules of execution. Each integration event presents unique challenges that must be tackled in addition to the use of standards. Focusing on integration best practices for interoperability is essential for designers to accomplish true interoperability within an event, but also coordination across other programs on the techniques can help interoperability across projects and organizations.

The aspect of interoperability that we will address in this paper is how to manage common expectations of each application's role within a larger perspective of the System of Systems (Jamshidi 2008) objectives. The applications' roles and how they meet those requirements are critical elements of interoperability that are often overlooked or overly simplified.

2.1 Applicability

This paper will use the M&S domain to illustrate interoperability complexities. In addition, the important points and lessons learned can be applied to any distributed computing environment (Metevier et al. 2010). More interestingly, the points made with regard to multiple, but otherwise independent, functions cooperating with a common understanding of the process and goals can be applied to people as well. Workflow processes executed by humans also require a shared perspective of the information shared, workflow objectives and processes wherein each individual resides. Developing the design and concept of operations for the Army's Mission Command networks and systems has the same interoperability challenges as distributed computing systems. Each element, whether human or computer system, must share well-structured information and execute capabilities within the context of a larger mission. Agent-based M&S is another example of multiple independent processes coordinating on a common framework. The applicability of lessons learned within the M&S domain can be applied to these and many more domains. The inverse is also true, that the M&S domain can learn many lessons of interoperability from other domains that have multiple application services cooperating for a larger goal.

2.2 Data Exchange

For applications to cooperate, the ability to exchange data is important. The information-sharing infrastructure can vary including shared memory or computer networks using Transmission Control Protocol (TCP) or User Datagram Protocol (UDP). The standards available for software systems to communicate are numerous and mature so this area will not be covered in detail.

An important factor for application interoperability is the technical dependencies that the data exchange mechanism applies to the information structure and the application synchronization. Many of the distributed computing middleware specifications allow for enforcement of synchronizing delivery order,

timing, persistent data ownership management, message segmentation and remote method invocation, all of which require careful technical design and planning for the applications involved.

One example is when middleware allows for data representation as a persistent object that can be updated by multiple applications. In this case, the concept of operations for an application changing an attribute must be considered when requiring other applications to subscribe to changes and react independently and in parallel to those changes. Having multiple applications change the same attributes results in undesirable behavior unless concurrent programming techniques (Ben-Ari 2006) are used to avoid problems. Suppose a logistics model is tasked to monitor the battery level in a vehicle and launch a recharge behavior once the level drops below a predefined threshold. If there are two models decrementing the battery level for the vehicle such as the engine model and the radio model, the battery level attribute should have a lock (mutex) to retain data integrity. Some middleware architectures provide an optional mechanism for locking attributes (ensuring singular ownership), but they do not ensure appropriate usage from an SoS design perspective.

The explicit and implied update frequency requirements are also key factors to the design. When design teams are focused on the information exchange, the timing required for the data exchanges can easily be overlooked. The purpose of the messages dictates the necessary timing of messages. Inside a single software application, programmers use direct notification design patterns (Gamma et al. 1994) to share information across threads or methods/functions. Within distributed middleware architectures, the available design patterns for notifications (Joseph et al. 2008) are limited. In particular, notifications between distributed applications are limited to what can be sent over a potentially unreliable network and require each application to properly interpret the logical meaning of each notification.

Most architectures follow anonymous publish and subscribe patterns. This allows the listener to notify the publisher that it is interested in a type of information, but there are no other details allowed. The listener cannot filter subsets of the information within an object or interaction in the data model, nor inform the publisher of the reason it is interested in the data and the urgency in which the data is necessary. This type of information must be captured within the systems engineering process and explained to each model developer explicitly. There is no technical enforcement of this criteria so this is an area of possible interoperability problems and a weakness in even the most well thought-out federation. Testing these types of design decisions is more involved than simply testing the data exchange interfaces. The testing process must examine the information within the messages to ensure that the filtering criterion is being met.

The system design must account for these types of issues and provide guidance and enforcement criteria beyond the mechanisms provided by the middleware or network infrastructure. These issues become exacerbated when applications are only partially compatible with middleware architecture specifications. The advanced capabilities of the architecture middleware frameworks, such as time management (Baker 1999), can be complex and difficult to test. System designers should provide detailed information to the development teams describing how each of the middleware architecture's capabilities will be used within the context of their event's architecture.

2.3 Information Exchange

The object model definition provides a necessary but limited representation of the meaning of the bytes transmitted over the middleware infrastructure. The bytes being exchanged between applications must be interpreted into their intended meaning. Object models, and in some cases the middleware itself, define encoding methods and data types, complex structures, transmission mechanisms (reliable, best effort, etc.), name spaces and in some cases, inheritance. Middleware standards often dictate the meta-model for what can be represented within an object model. In most modern middleware architectures, the object models are independent from the data-exchange specification and can be changed for each event as long as they are consistent across all of the applications. Also, the method for encoding and decoding the information must also be agreed upon by the applications, but this area is generally easily accomplished by development teams due to the wide use and availability of reusable software libraries.

A major challenge is that object models are not semantically rich representations of the domain to be represented. They are typically limited to data hierarchy, structures, types, sizes and in some cases include middleware transmission specifications, such as reliable or best effort. Terms that are overloaded (i.e., have multiple meanings) in the English language are often used. For example, a 'Tank' could represent the ground vehicle with tracks and a turret or a vessel, such as a gas tank. Imprecise terms are used such as entity, radio, sensor, etc. whereas the applications within the system have multiple types of the objects and more importantly each application has different fidelity (accuracy) and resolution (level of detail) representation of each of the objects.

The source of most language discrepancies comes from engineers trying to map their internal data structures to an existing object model. In many cases, the two data representations do not match up well and the engineer is forced to make assumptions or convert data between two incompatible data types. For example, two applications could represent damage in different ways (percentage of health as seen in game applications versus types of damage, such as communication, mobility, fires, etc.) while having no ideal way to map them. Each conversion potentially introduces errors to the data, e.g. rounding errors. The best way to avoid ambiguous or misleading data is to start design work from the top down. The functions, their interfaces and the workflow of the shared information should drive the object model rather than the inverse.

2.4 Information Semantics

Information semantics refers to the meaning of the data represented within the object model. It is difficult to ensure unambiguous language is used in the object model representation with accurate and explanatory comments. The object model comments should explain the format of the encompassed information, such as units of measure (meters, kilometers, miles, etc.) and resolution of terms when multiple interpretations are possible. An example of an ambiguous attribute is the orientation for a tank (and this time we mean the ground vehicle with tracks and a turret). Orientation could be where the tracks are facing and/or heading, but it could also be where the turret is pointed. The turret can turn independently of the tracks, and the tank can move both forward and backward, leaving the question of orientation if the tank is stationary. In the case where a mobility model and a fires model are both representing their respective portions of the tank, the difference obviously becomes critical. They each need the orientation of their respective tank parts which are independent of each other but there is only a single attribute. In this case, the object model needs to be adjusted to allow more specific fields.

Each model must also understand the purpose of sending or receiving information. In some system designs, the same message can be published in multiple ways and each can have its own purpose. For example, a Computer Generated Forces (CGF) model, which is made up of entities representing people, vehicles, etc., may be responsible for publishing SituationReport (SITREP) messages with the entity's status periodically in order for a situational awareness (SA) model to publish over the simulated network. The SA model may send the resultant communicated message in the same message format. Receivers, i.e. behavior models, of the message must understand why they have received each message and act accordingly. The behavior model should use the information within the SITREP to act on behalf of the publishing platform, but it should not use the information to act on the behalf of any other entity until the other type of SITREP is received (the one determined to be sent and received by other entities).

2.5 Function Roles Within System Composition

When applications are combined to accomplish a larger behavior, the applications become analogous to methods within a single application and to agents within an Agent-based M&S architecture. They must each implement their respective functionality within the scope of the larger context. In most cases, the reason for the SoS approach is to accomplish a complex behavior set. This implies that the problem space is difficult and each application implementing its function with well-defined borders and interfaces becomes even more critical.

Similar to systems engineering for a single software application, the systems engineering for a distributed simulation must capture high-level requirements and decompose how the system will operate down to the interfaces between functions. Systems engineering for typical distributed simulation events involves incorporating existing models to work together. Many systems engineers focus on the interfaces and the basic descriptions of capabilities of each application. Limiting the focus to these matters is the root of interoperability problems that rarely get noticed until systems are communicating and the data is analyzed.

The low-level interoperability problems of getting applications to share information, such as data structure inconsistencies, are easy to identify and in most cases, are easy to fix. They can be time consuming, using the budgeted integration and testing time to simply get bytes flowing between applications. The interoperability of business logic in the applications is more difficult to identify because it requires a detailed examination of the data along with the behaviors with a focus towards the intended high-level behaviors.

For example, if the goal is to drive a unit's high level behaviors from one model (Model A) while the entities' low level behaviors (e.g. moving and shooting) are represented within another model (Model B) to move and shoot, then the behaviors must be aligned. Most entity models have organic behaviors, like reacting to detection of enemy forces. If Model A's internal behaviors are to retreat when detecting an enemy force, then it will continually override all commands by Model B to move toward the enemy and engage them.

3 SYSTEMS ENGINEERING FOR INTEGRATING DISPARATE SYSTEMS

The goal of the systems engineer when integrating disparate functions should be to encapsulate functions into logically separable capabilities (Gallant and Gaughan 2010). The functions should be as modular as necessary depending on the larger system design and the need to compose multiple system configurations. The system designer's goal when attempting to facilitate future system reuse is to decompose the functions available from each application into building blocks that can be rearranged to accomplish multiple system designs, capabilities and execution behaviors.

The difficulty with this task lies in the details of the implementation within each model. A systems engineer should attempt to design a solution as if designing a system from scratch. This will force the systems engineer to consider details of the solution space that might not be considered if simply figuring out how to plug applications together. It is easy to assume technical implementation details based on an application's brochure or by skipping the detailed and thorough data semantics and workflow discussions. Making assumptions that a concept is interpreted the same way across applications is the cause of many interoperability issues that often don't get noticed until late in an event's life cycle.

This problem is further complicated within the military M&S domain due to the high rate of change that our military faces in the field and multitude of potential concepts that need to be represented within simulation. Simulation representations of warfare need to adjust faster than actual warfare in order for our military to use M&S for concept development, research, analysis, testing and evaluation, etc. As an example, ensuring that the many functions within the networked effects chain are all operating with the same understanding can become difficult when the military's process is often changing. Moreover, the applications are developed by different services or branches, there are multiple authoritative sources for the operational process and they have varying funding levels, which can make it difficult to keep up with the changes.

When attempting to encapsulate functions, we have found that most models have assumptions of their place within the larger behavior workflow. These assumptions can lead to undesired behavior when composed into a different workflow design based simply on the title of the behavior. For example, if an entity model receives a command from a Command & Control (C2) model, there could be many assumptions made. Problems arise when the models do not agree on all of the second order effects of the command. If the entity model has reactive behaviors that cause a retreat when an enemy entity is detected, the entity may never achieve the intent of the command from the C2 model, which could be ordering an attack

where a retreat directly conflicts with the intended high-level behavior. These types of behavior conflicts cannot be detected simply by agreeing on the object model / messaging between systems. The modeling goals and required overarching behaviors must be decomposed and analyzed with discipline and consideration towards the higher level SoS capabilities.

3.1 Cost Considerations in Systems Engineering

Cost considerations should encompass the required life of the technical solution, which in some cases may span multiple projects, programs or organizations. When designing the architecture for an event, the technical teams focus on the technical merit of each design decision. The program managers typically restrict the possible solutions to keep the execution within the available funding for that project.

The main cost driver for integration of many systems is the labor for software changes to the individual systems. In many cases, the systems were each developed for disparate reasons and constrained to specific technical architectures. The modifications necessary to migrate to a new architecture can be time-consuming and expensive.

Each project has finite funding and a limited set of objectives. The expense to migrate systems is duplicated across each project that has a unique architecture. If system architectures could be reused across programs, the total expense across the lifecycle of an application would decrease. If an application could reduce the interoperability modifications to save software development labor costs, the overall cost of M&S would decrease across the domain.

This is more difficult than it initially appears due to each environment having unique interoperability challenges. One standard or system architecture does not fit all problems. Cost and time are the limiters for any technical integration and must be considered. Finding the balance between the best technical solution and an affordable price is critical. One way to do this is to consider the main cost drivers in the technical design as decisions that need to be addressed. There may be multiple ways of accomplishing the same technical goal. Although there will likely be technical dependencies among design decisions that imply additional costs, understanding the technical dependency chains with regard to cost will help the design team make good interoperability design decisions while managing the cost to each project and ultimately to the community.

4 INTEGRATION APPROACHES

Many new approaches have been taken to address the classic interoperability challenges in recent years. These new approaches attempt to apply a new technology or technique to interoperability. Some of the new techniques were created for slightly different types of problems but are believed to be applicable to the distributed M&S interoperability problem space. This section will review three of these approaches' goals, advantages and disadvantages.

4.1 Developing a Canonical Object Model

Some projects have attempted to define a canonical object model that represents everything that may be represented across simulation environments. The goal is to provide translations from each of the targeted object models to the canonical object model. The basis for this approach is to allow translation from any target object model to any other target object model by translating the source data to the canonical representation and then translating to the destination object model. An advantage of this approach is to get the participating organizations to attempt to merge their data structures and underlying semantics into a single canonical object model. This would influence future design decisions within the participating simulation environments to adhere to the common object model. Over time, the simulation environments would likely become more interoperable due to the open discussion and common interoperability goal. However, there are numerous challenges with this approach.

4.1.1 Object Model Translation Errors

The most obvious drawback from this approach is the need to translate data twice between object models. Instead of translating a data structure from one object model to another object model, an intermediary object model is introduced forcing two translations instead of one.

The resolution between object models may be different, for instance when one object model has more detailed structures incorporated. Even simple conversion between units, coordinate systems and other varying ways of representing the same data can introduce basic mathematical rounding errors. There will also be inaccurate functional translations due to the lack of object model alignment. In some cases, the data structures will simply be mutually exclusive. If two object models decompose a concept similarly, yet in a significantly unique manner, then there is no easy way of translating between the two. Either business logic will need to be introduced in the gateway translation or one or both of the models will need to change how they represent the information.

4.1.2 Object Model Subsets

To avoid a double translation, the target object models can be used as parts of the canonical object model. The concept is that this removes the necessity of one of the translations because the canonical object model will be identical with one of the target object models. The fallacy with this approach is that the canonical object model needs to have a single representation of each concept to truly be a canonical reference. As the number of target object models increases, the likelihood is low that the canonical object model will still be identical to any of the given object models. If the concepts are not merged, but rather added together, it defeats the purpose of the common object model, which is to have a single authoritative data representation among the participating simulation environments.

4.1.3 Semantics and Operational Use Not Captured

Translating between object models cannot achieve true interoperability in most cases because object model interfaces are, and should be, specific to how a known set of components will cooperate. Once the models and purpose of the architecture changes, the object model must change accordingly in order to facilitate each exercise.

Simulation middleware architectures have unique means to represent their respective object models. These formats vary in their ability to capture semantically rich information along with the data structures. The object model limitations require system designers to capture the higher level systems engineering information and data semantics in a separate mechanism. These semantically rich descriptions are not as standardized within the M&S community as lower level middleware specifications and data exchange formats. Each military service, organization and program typically has their own way of decomposing warfare into representative functions. This implies that the underlying modeling may be significantly different and difficult to integrate.

4.2 Creating an Additional Layer of Behavior for Interoperability

Another approach is to create a layer above the existing legacy applications that can be manipulated according to the system requirements. A mechanism is built to orchestrate the behaviors of integrated models. A translation occurs where necessary between the added layer and the existing simulation model. The advantage to this approach is that it allows the user to design the execution without requiring changes to the simulations themselves. A drawback is the creation of additional overhead without solving the underlying interoperability issues of System of Systems M&S.

4.2.1 Limited Capabilities Across System of Systems

This approach limits the capabilities available at the orchestrated System of Systems behavior level. In order to execute SoS M&S, there is a need for a mutual understanding by the models of their role in the larger context. Some examples that this approach cannot handle include:

- Multiple models representing different portions of the same platform, which requires high frequency communication between two models. Interfaces must be opened up when the CGF is not originally developed for this use case.
- External behaviors commanding entities in a different application, which gets complicated when the CGF representing the entities has internal organic behaviors, especially reactive behaviors. Reactive behaviors can force the entities to contradict the intentions of the external behavior model.
- Modeling resolution differences existing between applications, which occurs due to the fact that models built for different purposes by disparate organizations with varying levels of funding will always have different representation of SoS common data objects.
- Performance data that effects behavior execution needing synchronization across applications, which requires a common understanding of the capabilities of the military systems in order to coordinate functions within a larger behavior.

4.2.2 Abstraction of Technical Details

This approach abstracts away the technical details of the integrated applications to ostensibly expedite and ease integration. But, this approach does not address the fundamental issues of interoperability so they can be addressed with each application. This approach is good for quick turn-around of scenario implementation without spending a lot of time and money making engineering changes to the models. The main limitation of this approach is the narrow breadth of the simulation capabilities available due to the complexity of true system interoperability. If an application simply does not allow an entity to behave in a certain way, then the abstraction layer is limited to only the available interfaces and behaviors.

Defining these types of constraints within the abstraction layer complicates the behavior representation and arguably makes the development and management of the abstraction layer more difficult, time-consuming and expensive than changing the applications to integrate directly. It also degrades the confidence in the Verification & Validation (V&V) of the overall system. In particular, each model may be independently verified and validated and have a pedigree for the purpose with which it was intended. When those models are integrated together, the V&V must be redone with an SoS perspective (how the systems are used for the broader system's goals). This approach of adding an additional layer invalidates all application pedigrees due to the additional layer of execution, translation and data management.

4.3 System Design Captured for an Executable Architecture

Another approach considers a systems engineering methodology and toolset for capturing a thorough set of linked distributed M&S architecture information. The goal is to link functional requirements to technical design while retaining traceability of the resultant data to the requirements and increasing semantic agreement among technical solutions. The design is initially captured functionally and then indirectly linked to applications and to a custom object model as appropriate. Tying requirements through design ultimately to the technical specifications, test cases and even execution forces a complete and up-to-date representation of the system design as it applies both functionally and technically. System requirements can be traced through design all the way to automatically generated test cases and data collection plans based on the data elements assigned to represent the high-level functional requirements in the design. Abstracting away the details of distributed M&S does not eliminate the need to manage them. This approach may give the impression that the complexity of integrating models is removed. Complex design issues still

need to be addressed, but the capturing of the design decisions in a way that provides modularity through encapsulation leads to more composable architectures.

4.3.1 Thorough Capture and Linking of System Design Information

The goal of this approach is to lower the barrier of entry for M&S to be used by facilitating quicker design, reusable system components and even automating many of the integration, testing and execution efforts. Distributed M&S is difficult and requires a large amount of skilled staff, which are hard to find. The complexities in setting up the M&S architecture both operationally and technically cause schedule slips and expensive adjustments and setup in the technical solutions.

The flexibility required to implement this goal requires system functionality encapsulated into appropriately sized pieces to be arranged into larger capabilities as needed with as little engineering effort as possible. A mature implementation of this approach could provide an architecture robust enough for decision-oriented analysis, while maintaining flexibility and quickness, as well as provide traceability necessary for compliance with U.S. Army-grade Verification and Validation guidance (M&S CO VV&A 2006). This could save the Army tremendous amounts of time and effort when constructing distributed M&S environments for various uses.

In this approach, the system design is captured in modular, functional blocks with well-structured interface definitions. This results in better composability of simulation events by only selecting the functions required per the event's objective. This approach transforms the methodology for developing a distributed simulation environment from jamming models together, to being able to carefully select functions and have the technical details of the implementation abstracted away from the designer. This creates a 'buffet line' of simulation capabilities for developing distributed simulation environments allowing the user to pick and choose specific capabilities. The system manages the operational and technical dependencies for the user and resolves the necessary models, object model elements and architecture criterion.

This builds upon the technical abstraction by capturing, linking and navigating more details to allow actual implementation of the distributed simulation architecture without human involvement. All the information necessary to execute the architecture is stored and important technical design elements are linked as appropriate, such as scenarios with functions, functions to applications, software configurations to implementation choices, etc.

4.3.2 Capturing Thorough Design and Execution Details Facilitates Automated Execution

Capturing design information in this way allows an increase in the use of generative programming techniques (Czarnecki and Eisenecker 2000), or automatically generating executable computer programming artifacts from a higher level source, in order to quickly deploy a SoS architecture for military analysis. Automation saves time and money while promoting robustness, flexibility, repeatability and traceability.

The interfaces, functional requirements and business logic are all captured in the context of the larger system design. The generated applications and technical artifacts are all based on the captured architecture design. Normally, test cases, data collection plans and surrogate applications are developed manually taking more time (more cost) and introducing possible errors. Automatically generating technically detailed products ensures more accuracy, repeatability and speed, which all result in more cost-savings.

The generation of the test federates can be expanded to include more complex execution behaviors, such as the generation of working code blocks within the generated federates from pseudocode. This code generation, along with separation of business logic from middleware specification details, allows rapid generation of design, tests, surrogates and even a mechanism to rapidly create a model that works across many middleware architectures (HLA, Test & Training ENabling Architecture (TENA), etc.) and is already captured within the system design.

Automatic deployment and execution of the SoS architecture can be achieved due to the capture and organization of all of the key data for executing the distributed simulation environment in addition to the

design knowledge. understanding operating system, memory and networking requirements for each model allows for a virtual deployment on managed computing assets. The goal is to capture all of the information necessary within a structure that is as composable as possible in order to allow easy rearrangement. The system execution details are then tied to the functional goals of the user. Capturing the linkage between high-level event objectives and the detailed execution information for each of the included systems allows for functionally purposed execution of a system architecture. Once capabilities are mapped to technical implementation and launching the system is automated, an analyst can explore the system design to dictate the scenario used based on metadata such as force representation, warfare functions executed, terrain types and many more metadata attributes. The analyst can launch the execution of the event through a user interface and the collected data can be directly tied to the system requirements because the object model elements are captured within the design.

5 SUMMARY

Integrating disparate systems is complex and requires intelligent use of software development practices. It is akin to reusing software libraries. Software developers must encapsulate functionality while retaining appropriate interfaces that allow the functions to be orchestrated together for the achievement of the larger system requirements.

The core issues with system interoperability hold true no matter how one approaches the problem. Projects have attempted to approach interoperability in many ways using the newest technologies and software practices. While applying new technologies to solve interoperability is interesting, the fundamental keys to interoperability must be considered first and foremost. The event's objectives and context of the systems cooperating must be well described and understood. Each system's role should be understood and decomposed, including the model representation, information exchange semantics, data models and data exchange (including frequency, control and temporal synchronization).

Integrating multiple applications to provide a greater capability can seem like simple arithmetic. By adding an Unmanned Aerial Vehicle (UAV) model with a new sensor model, one might think that the end result will allow for UAVs to incorporate the new sensor model for analysis. However, without careful consideration, the two models may not be able to easily communicate, exchange data, coordinate temporally appropriate information and cooperate together functionally and within the context of the system's architecture requirements.

Careful consideration must be applied to the interoperability issues described in this paper to determine the feasibility and final capability of the integrated systems. This feasibility analysis should be executed early in the decision process and the cost of integration based on the interoperability spectrum should be a major consideration of the systems engineering effort. In some cases, creating or improving existing models may be a better overall approach than integrating two seemingly compatible models that after detailed review, simply don't have the proper functionality to be truly interoperable.

When implementing a distributed simulation environment, the longer term benefits of capturing and linking the systems engineering information are particularly important for future reuse. Tackling interoperability challenges in a more generic way will allow for better functional and technical encapsulation. This can lead toward more frequent and cheaper reuse of models and simulations across organizations and programs. Moreover, capturing the systems engineering information in a discrete and linked method allows for changes in requirements to easily be traced through the design facilitating better understanding of the impacts of changes (both required and unintentional).

REFERENCES

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

Baker, T.. 1999, March. "Time Management in Distributed Simulation Models." *SimTecT Simulation Conference*, Melbourne, Australia.

Ben-Ari, M.. 2006. *Principles of Concurrent and Distributed Programming*. 2nd ed. Addison-Wesley.

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 Version 2.0.* Available via http://cio-nii.defense.gov/docs/DoDAF%20V2%20-%20Volume%201.pdf.

Department of Defense (DoD) Modeling and Simulation Coordination Office (M&S CO). 2006. *VV&A Recommended Practices Guide.* Available via http://vva.msco.mil/Default.htm.

Gamma, E., R. Helm, R. Johnson and J. Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. 1st ed. Addison-Wesley Professional.

Gallant, S. and K. Snively. 2008. "Effective Distributed Simulation Design for Cross-Domain Interoperability." *Simulation Interoperability Workshop*, Providence, RI.

Gallant, S. and C. Gaughan. 2010. "Systems Engineering for distributed live, virtual, and constructive (LVC) simulation." In *Proceedings of the 2010 Winter Simulation Conference*, edited by B. Johansson, S. Jain, J. Montoya-Torres, J. Hugan, and E. Yücesan. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

IEEE Xplore Digital Library. 2010. *1516-2010 IEEE Standard for Modeling and Simulation (M&S) High Level Architecture (HLA) – Framework and Rules.* Available via http://ieeexplore.ieee.org/xpl/freeabs_all.jsp?arnumber=5553440.

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

Joseph, J., J. Hogg, D. Ossipov, M. Mascaro and D. Garber. 2008. "Architectural Patterns for Distributed Computing." *The Architecture Journal*, 17, 8 – 15.

McCray, P. and K. Snively. 2008. "Functional Component Testing for Distributed Simulations." *Simulation Interoperability Workshop*, Providence, R.I.

Metevier, C., C. Gaughan, S. Gallant, K. Truong and G. Smith. 2010. "A Path Forward to Protocol Independent Distributed M&S." *Interservice/Industry Training, Simulation and Education Conference (I/ITSEC)*, Orlando, FL.

Page, E. and D. Lunceford. 2001. "Architectural principles for the U.S. Army's simulation and modeling for acquisition, requirements and training (SMART) initiative." In *Proceedings of the 2001 Winter Simulation Conference*, edited by B. A. Peters, J. S. Smith, D. J. Medeiros, and M. W. Rohrer, 767-770. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

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

Simulation Interoperability Standards Organization (SISO) Product Development Group (PDG) for the Core Manufacturing Simulation Data (CMSD). Available via: http://www.sisostds.org/StandardsActivities/DevelopmentGroups/CMSDPDGCoreManufacturingSimulationData.aspx.

Simulation Interoperability Standards Organization (SISO) Product Development Group (PDG) for the Military Scenario Definition Language (MSDL). 2011. Available via http://www.sisostds.org/StandardsActivities/DevelopmentGroups/MSDLMilitaryScenarioDefinitionLanguage.aspx.

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

Subramaniam, V. 2008. *Programming Groovy: Dynamic Productivity for the Java Developer*. Pragmatic Bookshelf.

Tolk, A. and J. Muguira. 2003. "The Levels of Conceptual Interoperability Model." *Simulation Interoperability Workshop*, Orlando, FL.

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

Wampler, D. and A. Payne. 2009. *Programming Scala: Scalability = Functional Programming + Objects*. 1st ed. O'Reilly Media.

AUTHOR BIOGRAPHIES

SCOTT GALLANT is a Systems Architect and Software Engineer with Effective Applications Corporation. He has over fifteen years experience in distributed computing including Army modeling and simulation. He has led technical teams on large U.S. Army programs for distributed software and federation design, development and execution management in support of technical assessments, data analysis and experimentation. He is currently the lead Systems Engineer and System Architect on the MATREX program managed by the U.S. Army Research Laboratory (ARL) Simulation & Training Technology Center (STTC). He earned his Bachelor of Science in Computer Science from George Mason University in Fairfax, VA. His email address is <scott@effectiveapplications.com>.

CHRIS GAUGHAN is the Advanced Distributed Simulation Environments Chief Engineer and Deputy Technology Program Manager of the Modeling Architecture for Technology, Research & EXperimentation (MATREX) program at the U.S. Army Research Laboratory (ARL) Simulation & Training Technology Center (STTC). He has a diverse portfolio of distributed simulation projects that support the full spectrum of the Department of Defense Acquisition Life Cycle. From 2004-2009 he worked at the Edgewood Chemical Biological Center (ECBC), where he served as the Configuration Manager of the Chemical-Biological-Radiological-Nuclear Simulation Suite. During his tenure at ECBC, he was 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>.