Abstract

Interoperability of constituent systems is the primary challenge for a large-scale distributed Live-Virtual-Constructive (LVC) simulation activity. The grand challenge is to integrate, for the first time, all aspects of interoperability necessary not only to enable but to ensure interoperability throughout an large-scale LVC system. Previously, a handful of these aspects of interoperability have been partially addressed in isolation. Only an integrated solution of all aspects of interoperability into a computer-enforceable architecture will make possible attaining the goal of ensuring interoperability throughout a large-scale LVC activity.

The enforceable nature of the LVC activity models implies that the models accurately reflect the “reality” of the LVC activity. Without enforcement, the complex nature of the myriad of components would tend to introduce inaccuracies into models, ultimately rendering those models useless. Actual enforcement of the models elevates the LVC software systems’ usefulness to a level that would otherwise be unattainable in practice.

Preparing for an LVC activity involves answering deceptively simple questions such as, “Will an individual piece of software work during this LVC activity?” At present, reliably answering such questions is impossible, because no detailed, precise, and accurate descriptions of the software or the LVC event exist. Other simple questions such as, “Which data collection systems are suitable for use in this LVC activity?” are equally difficult to answer reliably.

This paper discusses the need for computer-enforceable models of individual software components as well as entire LVC activities, along with creating new extensions to the existing TENA object modeling language and to the TENA Middleware itself.

1. Introduction

Programming large-scale distributed and real-time simulations presents significant challenges. Considerable research attention has been focused on the fundamental issues involved in the creation of such simulations, and with good reason, given their inherent complexity. Nevertheless, one important class of large-scale distributed and real-time simulation systems has, thus far, not received attention from the research community commensurate with its complexity: live, virtual, and constructive (LVC) systems.

LVC systems combine three types of distributed simulations and applications into a single distributed system. These three types are:

- **Live**—Distributed applications to support testing and training with real, physical assets. These assets typically include soldiers, airplanes, tanks, ships, and weapon systems.
- **Virtual**—Distributed simulations to create simulators of physical assets which support testing and training in a virtual environment, e.g., airplane simulators, tank simulators, etc.
- **Constructive**—Distributed simulations to create synthetic environments in which simulation models of physical assets are used for testing and training—primarily of procedures, doctrines, command, control and communication, “what-if?” scenarios, and predicting the effects of simulated events such as an explosion.

The goal of combining these three types of distributed systems is to present real, virtual, and synthetic assets into one seamless and coherent environment operating in real-time.

2. The Nature of LVC Systems

LVC systems present challenges not found in other real-time distributed simulation systems. Software architectures and paradigms originally developed to support interconnecting systems for distributed simulation are not necessarily well suited to support the live component of LVC systems. This is due to the fact that when real, live systems are mixed with virtual reality and/or constructive simulations, the demands of the live systems dominate the resulting LVC system.
Creating an LVC system is a complex multidisciplinary undertaking. The expense of establishing and maintaining the capability to create and simultaneously operate simulations, simulators, and real physical systems may, in part, contribute to their relative lack of attention from the research community. In fact, it may well be that few endeavors require the cost and complexity involved with LVC systems.

3. The Use Case for LVC Systems

At least one substantial endeavor makes extensive use of LVC distributed simulations and applications—the testing and training activities of the U.S. Department of Defense (DoD).

The U.S. DoD ranges and laboratories use systems of sensors to take measurements for the purpose of testing and/or training. Many of the these sensor systems are embedded systems. The testing and training events occur in the real world, meaning that real missiles are launched, real tanks are driven, real planes are flown, etc.; and so real measurements must be taken in real-time and with high-performance (e.g., latencies measured in milliseconds). The sensor systems are themselves inherently distributed, typically over a large geographic area. The sensor systems can include half a dozen to several hundred individual component sensors. So, DoD ranges are large-scale, distributed, real-time, and embedded (DRE) systems.

For various testing and training activities, these large-scale DRE sensor systems utilizing live data are augmented with simulator systems and simulations yielding an LVC system. Such an LVC system can be used, for example, to enable a pilot in a real airplane in Nevada to fly with a pilot in a simulator in Florida on a mission to engage synthetic tanks that are being simulated in Texas.

4. Large Scale LVC Systems Present Unique Interoperability Challenges

Individually, the parts of a large-scale LVC system such as those employed by the U.S. DoD test and training ranges and laboratories are quite complicated. When these individual parts are combined to form an LVC system as a whole—the result is a large enterprise made up of a variety of complex interrelated systems.

There are many aspects of the large-scale interoperability of the systems comprising such an enormous enterprise, any one of which, if overlooked, could impair or even cripple the functioning of the entire enterprise. One aspect of interoperability is providing common low-level communication protocols, enabling communication among systems in the enterprise. A second aspect of interoperability concentrates on commonality, or data agreement, about the nature and form of data exchanged.

Over the years, industry and academia have produced a variety of protocols and middleware products addressing one or both of these first two aspects of interoperability, such as Distributed Interactive Simulation (DIS) [1] and the High-Level Architecture Runtime Infrastructure (HLA/RTI) [2]. The U.S. DoD testing and training ranges and laboratories have made use of both DIS and HLA/RTI as part of their approach to creating LVC systems. However, simulation software architectures such as DIS and HLA/RTI were conceived primarily to support constructive simulations. As such, they are ill-suited to support the demands of LVC systems (in particular, the live component of LVC systems).

More recently, these U.S. DoD facilities have turned to the Test and Training Enabling Architecture (TENA) [3] [4] [5] to support their LVC activities. TENA has addressed the first aspect of interoperability by building upon the existing body of work on low-level data communication protocols [6] [7] [8] [9]. TENA has focused on the second aspect of interoperability by founding the fundamental concepts of its architecture on formal computer-enforced data format agreements [3] rooted in the model-driven software principles of Model Driven Architecture (MDA) [10].

Yet even satisfying these two aspects of interoperability (common protocols and data agreements), has proven to be insufficient to ensure interoperability throughout an enterprise as large and complex as an LVC system. Common protocols and data format agreements are only the first steps along the path to interoperability. Understanding the next steps requires insight into how the software systems and applications used in large-scale LVC system are developed, deployed, maintained, and described.

5. Large-scale LVC Systems are Developed in a Decentralized Manner

It is possible to create a large-scale distributed and real-time simulation made up of a particular version of a specific application that is developed by a single development team. When developed in that way, such large-scale distributed and real-time simulations completely avoid tremendous complexity and barriers to interoperability.

By their very nature, LVC systems cannot be composed a single application homogeneously used throughout the entire LVC system. This is a result of the wide differences in the nature of the applications that support live systems, the applications that create virtual reality simulators, and the applications that create constructive simulations.

When LVC systems are used at large-scale, as is commonly done in the U.S. DoD testing and training community, the nature of the large-scale tends to further increase the number of distinct applications used throughout the entire LVC system. Given the large
number of applications in use over a large-scale LVC system, it is highly unlikely that a single software developer, a single software development team, or even a single software development organization developed every application.

Instead, it is nearly certain that multiple organizations, teams, and individuals contributed to various parts of the large-scale LVC system as a whole, often at disjoint periods of time. Thus, the development of these applications is itself performed in a distributed fashion—distributed both geographically and temporally. In addition, the development is nearly always decentralized, lacking a common authority to provide a uniform and consistent development process.

Achieving interoperability between the myriads of complex pieces of software used in LVC systems in the face of such a chaotic software development environment is extraordinarily difficult. Yet, the success or failure of the LVC system rests squarely with the ability of these software pieces to successfully interoperate.

Formal, computer-enforced agreements describing the nature and form of data exchanged in a large-scale LVC system are necessary to provide a common understanding of how and what data is to be communicated and furthermore, to ensure that that understanding is then implemented in every application comprising the system.

Without the use of formal, computer-enforced agreements detailing the nature and form of the data exchanged, it is extremely difficult to ensure that the applications in an LVC system are abiding by those agreements. The magnitude of this difficulty is increased because of the distributed and decentralized development of the pieces of software used to create these LVC systems. TENA provides (and requires the use of) a mechanism to create these formal, computer-enforced data exchange agreements. The commonly used distributed simulation software architectures, DIS and HLA/RTI, have no such mechanism.

6. LVC Systems Require More than Just Common Communication Protocols and Data Exchange Agreements

The inherent complexity of large-scale LVC systems means interoperability cannot be achieved without a common understanding of the nature of the systems and the data exchanged among them. However, having such understanding does not, in and of itself, result in interoperability. To achieve interoperability, this understanding must transform into a sort of “contract,” enforced everywhere throughout the enterprise.

The complexity of such a large enterprise makes guaranteeing these contracts a practical impossibility for humans—but not for computers.

To gain an appreciation of the scope of the interoperability challenges faced by LVC systems, consider the simple (and admittedly contrived) data exchange agreements shown in Figures 1 and 2. Suppose a constructive simulation that is to be used in an upcoming LVC activity is using a 2-dimensional representation of an EnemyLocation message like the one shown in Figure 1.

![Figure 1. A very simple data exchange agreement for a 2-dimensional EnemyLocation message shown in TENA definition language (TDL).](https://www.tena-sda.org/)

Suppose further that a simulator system to be used in the same upcoming LVC activity is using a 3-dimensional representation of an EnemyLocation message like the one shown in Figure 2.

![Figure 2. A 3-dimensional version of the EnemyLocation message shown in Figure 1.](https://www.tena-sda.org/)

This (overly-simplified) example brings to light the precisely the kind of obstacles facing those seeking to create large-scale LVC systems. The simulation using the 2-dimensional EnemyLocation message and the simulator using the 3-dimensional EnemyLocation message may each perform flawlessly in isolation. But when combined into a single LVC system, suddenly an unexpected and potentially disastrous interoperability problem arises.

This type of interoperability failure is unlikely to occur if all the software applications are written by a single, well-coordinated software development team. But, as discussed above, it is a practical “impossibility” to create a large-scale LVC system in that manner. Realistically, the best that can be hoped for is that the software development teams creating the various pieces of software used in the LVC activity are communicating with each other. And often, even that is simply not possible.

Naturally, being in communication by no means guarantees that the products of the software development teams will be interoperable. In the example described above with the simulation using the 2-dimensional EnemyLocation and the 3-dimensional simulator, each software development team would...
answer “Yes” to the question “Does your software use the EnemyLocation data agreement?”

It should be observed that the simulation and the simulator in this example can be assumed to be using identical communication protocols. In fact, it can be assumed that they are even using identical middlewares, e.g., HLA/RTI, OMG Data Distribution Service [11], or the TENA Middleware. None of middlewares or the communication protocols they employ would provide interoperability between the simulator and simulation described in this example.

However, one of the middleware systems listed above would, in fact, detect the interoperability problem before allowing the LVC system to be started up: TENA. TENA is designed to be both easy to use and, more importantly, hard to use wrong! A bed-rock principle of TENA is to define interoperability contracts that are then enforced by the computers. One such basic contract is that there can only be one definition of an EnemyLocation message. In this example, the TENA Middleware detects that the simulation and simulator are in violation of that contract.

While the example discussed here is greatly simplified, the type of interoperability failures it exemplifies happen far too frequently with LVC systems. Worse still, for LVC systems not using TENA, these types of interoperability failures may silently occur, creating failures that are not always easy to detect or correct.

7. What More Technology is Required to Reliably Create LVC Systems?

LVC systems succeed or fail on how well their constituent parts can interoperate. Interoperability can only be assured when there are formal computerized contracts detailing

1. the characteristics of the of the individual systems,
2. the data exchanged among them, and
3. the nature of the enterprise activity as a whole.

These contracts must then be computer-enforced to ensure that they are actually followed throughout the enterprise.

TENA has done considerable work on the enforcing data exchange contracts, but significant research into the means to accurately characterize individual systems as well as the nature of an entire LVC activity as a whole, remains, along with the research into how best to enforce the characterization contracts once they are created.

The existence and enforcement of these contracts would, for the first time, make it possible to address the fundamental interoperability question that plagues those that need to perform LVC activities: “Will this software system interoperate within that particular activity?” in a way that assures the correctness of the answer.

These technologies will provide invaluable capabilities to support the successful planning, execution, analysis, and history of LVC activities, such as test events and training exercises for the U.S. DoD.

7.1. Enforceable Component Metamodel

A component metamodel provides a framework in which details about the constituent parts of a system can be described. The extent of the components of a test event or training exercise include things like software applications, simulations, stimulators, software libraries, documentation, use cases, configuration files, etc. The component metamodel defines the allowable breadth and scope of the details that describe the components. For example, a component metamodel would likely allow a version number to be assigned to a component. It would also likely allow one component to depend on another, allow a component to describe its intended purpose (e.g., “Logs position data”), describe its restrictions (“Only works with Windows XP”), etc.

The enforceable nature of the component metamodel implies that the component models created using the component metamodel will accurately reflect the “reality” of the component. One simple example of this enforcement would be to have the TENA Middleware verify that the operating system on which the component is being run matches the required operating system specified in the component model. Without enforcement, the complex nature of the myriads of components would tend to introduce inaccuracies into the component models, ultimately rendering those models worthless. Actual enforcement of the models elevates their usefulness to a level that would otherwise be unattainable in practice.

The descriptions of the software components of U.S. DoD test events and training exercises can be used in many ways. For example, during planning, descriptions of available components can be used to identify compatible components (e.g., hardware in-the-loop facility interfaces, instrumentation systems, loggers, viewers, simulations, etc.) to be used in the event. During execution, the component descriptions can be used (e.g., by the TENA Middleware) to enforce that the desired components are in use. During post-execution analysis, the component descriptions form part of record of what was done.

7.2. Enforceable Activity Metamodel

An activity metamodel provides a framework in which details about a given test event or training exercise can be described. The details of the activity would likely include descriptions of the particular U.S. DoD ranges participating in the activity, descriptions of the computers and instrumentation systems in use, the types of networks interconnecting them, the com-
ponent descriptions of the software applications used, and so on.

An activity metamodel enables the creation of detailed and precise, computer-friendly descriptions of test events and training exercises: past, present, and future. Such descriptions, i.e., activity models, are necessary but not sufficient to satisfy a great many needs. Examples include such needs as using an activity model as part of the result of the planning process, using the activity model during the exercise or event to compare the actual exercise or event to the plan, and after the activity to provide a historical record of what was done. As with the enforceable component metamodel, described in Section 7.1 above, the enforceable nature of the proposed event metamodel comes with new dimensions of technological complexity, yet also offers the promise of pioneering advancement to interoperability for complex enterprises.

As described above, there are numerous ways that activity models interact with test scenarios. Consider one scenario where an important enhancement to a critical data logging application is made. How can there be assurance that the new version of the logging application will interoperate with an upcoming event activity? Is the event going to be using TENA Middleware Release 5 or 6? Which version of the TENA Middleware does the enhanced logging application require? Suppose the enhanced logging application only runs on Windows Vista. How is a determination made if there are any computers running Windows Vista available in the event? These questions are just a tiny sampling of the kinds of questions that can be quickly and accurately answered with a formal, computer-understandable component model of the logging application and a formal, computer-understandable model of the event activity.

Developing this enforceable activity metamodel will involve ensuring that the activity metamodel itself is expressive enough to capture all the details necessary to describe the systems, software, networks, and data flows with enough accuracy and precision as to ensure interoperability. Developing the ability to enforce the activity models will involve augmenting the TENA Middleware to interpret an activity model and compare it with the “ground truth” of the actual execution of the activity described by the activity model.

7.3. Component Repository

A component repository provides a means to store components so that they are available for use in the enterprise. More than simply providing storage, the repository understands the component metamodel so that it can reason about the components it contains. This reasoning allows for user interactions with the repository such as navigating all the interdependencies of a component to verify availability, responding to queries about which components have a given capability, as well as tracking usage of components.

Currently, the U.S. DoD test and training community does not have a uniform or centralized warehouse containing information about what software components (e.g., tools, applications, simulations, libraries, etc.), or previous event configurations, are available and how to obtain them. Not only would a component repository address this need, but by reasoning about the model for each of these components, the component repository would be able to satisfy interoperability needs by supporting queries such as, “What data loggers are available that work with Release 6 of the TENA Middleware?” With an understanding of the activity metamodel, the component repository would be able to answer queries about software components’ availability to work with a given test event or training exercise. At present, no such system is available to planners or the engineers supporting the activities.

7.4. Software Component Build and Test System

As a rule, any complex system requires maintenance. The software systems used to support U.S. DoD test events and training exercises are certainly no exception. The nature of software maintenance implies that the software will be changed. Changes occur for a variety of reasons: bug fixes, security fixes, feature enhancements, capability extensions, and so forth. When a previously built piece of software (i.e., “a component”) changes, it must be re-built. Responsible software development practices require that the re-built software component be tested: first in isolation, then with regard to previous versions of itself (regression testing), and finally in conjunction with the other software components with which it interacts. This build and test cycle can be a time consuming, laborious process.

A software component build and test system would facilitate the development of the software components used to support test event and training exercise, such as loggers, viewers, gateways, software libraries, simulations, etc. A component build and test system would semi-automate the maintenance and testing of these software components by performing automated regression testing and stress testing of the software components. This system would be able to understand the interdependencies of the software components it builds to ensure that a change to one component is tested against all components affected by the change.

Employing this system will reduce the time, effort, and cost of developing and maintaining the software that supports U.S. DoD test events and training exercises. Perhaps more importantly, interoperability needs will be addressed by providing the means to obtain a build of a software component that was otherwise unavailable but necessary for some activity, e.g., a build of a software component such as a data logger for new version of an operating system. By
providing a means to perform extensive testing of software changes, interoperability failures that arise today due to unintended consequences of even small software changes can be detected early, and hence prevented from adversely affecting an activity.

In addition to this, the combination of the repository and the build and test systems will provide the means to enable semi-automated software source code updates that are often required for a given software component to take advantage of new versions of dependent sub-components. Such updates are a critical part of maintaining the considerable investment the U.S. DoD test event and training exercise community has in software. At present, this maintenance is often delayed or even deferred indefinitely due to the time, cost, and complexity burden imposed by current practices. The proposed repository and build and systems, along with the deployment system described below, would dramatically reduce this maintenance burden.

The proposed software component build and test system has a great many uses throughout the enterprise. As a simple example, consider an activity involving a pair of instrumentation systems, System A and System B, which need to exchange position data during the activity. If System A measures position in geodetic coordinates and System B measures position in geocentric coordinates then some software component must perform the coordinate conversions so that System A and System B can communicate.

When a new version of that component is needed (e.g., due to a bug fix), it must be built for all the computer platforms needed for both systems. Furthermore, the newly built component must be tested to ensure that changes have not introduced a new problem. In addition to this, any software components that depend on the coordinate conversion component (e.g., data loggers, data viewers, etc.) may themselves have to be rebuilt using the new component and re-tested for unintended consequences, in a sort of “ripple effect.”

This ripple effect plays a significant role in hampering the advancement, modernization, interoperability of systems in large enterprises. Advancement and modernization is hindered because the time, effort, and cost involved in smoothing out all these ripples is big enough that it may not be thoroughly completed throughout all the necessary software systems of a given activity. In the event that the ripples of re-building and re-testing are not thoroughly completed, unanticipated consequences of the change can impair interoperability.

The software component build and test system discussed here would greatly reduce or even entirely eliminate the burden of the re-build and re-test process described in this example. This result will be possible because of the supporting capabilities provided to the component build and test system by the component metamodel and the component repository.

7.5. Software Component Deployment System

A software component deployment system provides the means to (semi-)automatically install new software components and to apply upgrades, enhancements, bug fixes, and security patches to those software components to the computer systems in use throughout large enterprises.

Every large enterprise needs to be able to deploy software used to support the performance of its activities onto the appropriate computer systems. Inevitably, updates to previously deployed software components need to be applied. Currently, no comprehensive system exists to automate or even semi-automate the initial installation or the subsequent updating of the software used to perform activities in the U.S. DoD test and training enterprises. The software component deployment system discussed here would satisfy this need.

The proposed software component deployment would typically be employed during the preparation phases of an activity. After having described the planned activity using the activity metamodel, the software engineers provide the activity model to the software component deployment system. The deployment system extracts the pertinent information about what software components are required for each computer system called out in the activity model. Next, the deployment system interacts with the component repository to obtain the necessary software components. This step may possibly involve interacting with the software component build and test system to build some or all of the desired software components from source if no appropriate build was already in the repository. Once obtained, the required software components are installed on the computer systems specified in the activity model.

The goal of the software component deployment system is to completely automate the process outlined above. That is, the goal is to use an activity model created for a particular U.S. DoD test event or training exercise, so that a simple “push of a button” provides assurance that either all the computer systems specified in the event model have all the appropriate versions of software components installed, or alert the testers, identifying any systems not satisfying requirements described in the event model. In this way, potential interoperability failures can be detected and addressed—well before the activity is executed.

7.6. TENA Data Format Agreement Extensions

Existing TENA data format agreements define the form of the data exchanged, but not the nature of the data. The TENA data format agreements comprise what is essentially the “language of TENA.” As such, the more expressive and precise that language is, the
better it is able to precisely describe the complex communication exchanges of an LVC activity. For example, TENA can enforce an agreement that the speed of an article under test be represented as a 64-bit floating point value.

However, TENA’s data format agreements fall short of stipulating several critical characteristics of the nature and form of the data exchanged in large enterprises. More work is needed to fill the interoperability gaps caused by these shortcomings.

Extensions to these agreements could also enforce such things such as units (e.g., km/hr vs. mph) and constraints on the allowable range (e.g., between 0 and 30,000 km/hr).

Data format agreements are fundamental to enabling interoperability for LVC activities. The more detailed and precise an agreement, the less likely that an unanticipated gap in that data format agreement will lead to miscommunication, and hence a failure of interoperability. Thus, TENA data format agreement extensions will plug “gaps” that can lead to miscommunication.

The proposed TENA data format agreement extensions would typically be used during the preparation phases of an LVC activity, such as during the development of applications that support the activity. For example, applications using the TENA data format agreements as they exist today can exchange 64-bit floating point numbers labeled, “speed,” with a value such as “7.2.” The proposed TENA data format agreement extensions could, for example, further clarify that data exchange as 7.2 in a class of things categorized as “meters per second” thereby eliminating a source of miscommunication. TENA data format agreements are computer-enforced, thus errors are detected at the time of application development, which is well before the actual LVC activity occurs.

7.7. Programmable Adaptive Filtering

Release 6.0 of the TENA Middleware introduced the ability to filter data on the send-side, based on integer “tags.” For example, a data publisher can associate an integer “tag,” such as “7” to a particular piece of data. Similarly, a data subscriber can request data with certain “tags,” such as “7, 8, or 9.” Using these integer tags, the TENA Middleware can squelch a data transmission before it ever uses valuable network resources if the tag of the data being published is not in the subscriber’s list of desired tags. While, this tag-based filtering of Release 6 of the TENA Middleware is an improvement over the purely type-based filtering of previous releases, it falls short of what is truly needed.

Significant enhancements to the send-side data filtering capabilities of the TENA Middleware can be made by inventing mechanisms to allow TENA application programmers to define complex transmission criteria to associate with the data exchanged in LVC activities. These criteria enable the TENA Middleware to adapt to the data exchange requirements programmed by the TENA application developers, and thereby, make decisions that support interoperability while reducing the consumption of valuable network resources.

LVC activities often involve the exchange of a great deal of data. The nature of some LVC activities requires certain applications receive a restricted amount of the total amount of data available. Many reasons for restricting the amount of data an application receives are possible, such as constraints on the available network resources, constraints on the amount of data an application can process before failing under the load, or perhaps constraints from the LVC activity itself whereby a particular application is supposed to receive only a subset of the data to evaluate some test or training criteria. Whatever the reason to restrict the total amount of data, the ability to do so is a fundamental LVC need.

As a simple example, consider an application capable of processing only a certain number of radar tracks per second. A subtle and pernicious interoperability failure can occur if those restrictions are not followed, e.g., the application could fall behind or even crash. Working in conjunction with the component model for the application, and the event model for the LVC activity, the proposed programmable adaptive filtering support could support limiting the number of tracks received by the application.

8. Conclusion and Summary

The inherent nature of the software and systems used to carry out LVC test events and training exercises implies that such enterprises are composed of multiple heterogeneous systems, applications, and hardware platforms, geographically distributed at multiple sites. The large-scale distributed nature of these enterprises tends to further increase the number of distinct systems involved. Given the large number of diverse systems, it is certain that multiple organizations contributed to various parts of the enterprise at different periods of time. This development is nearly always decentralized, lacking a common authority providing consistent development processes and design decisions for each system.

When real, live systems are mixed with constructive and/or virtual reality simulations, the demands of the live systems dominate the resulting LVC system. Software architectures intended to support the creation of distributed simulations (e.g., DIS and HLA/RTI) are not particularly well suited to meet the unforgiving demands of live distributed systems.

TENA is a software architecture primarily used by the U.S. DoD testing and training community. The needs of that community run the gamut of large-scale, real-time, distributed and embedded software systems.
The applications used by that community are typically developed in a decentralized and loosely coordinated manner. Oftentimes, these applications are combined to form large-scale LVC systems.

The TENA Middleware is designed to enable the rapid development of real-time distributed applications, such as LVC systems. The TENA Middleware combines distributed shared memory, anonymous publish-subscribe, and model-driven distributed object-oriented programming paradigms into a single distributed middleware system. This unique combination provides a programming experience that is both simple and powerful for a broad variety of real-time distributed applications.

Despite all the advanced interoperability features incorporated into the TENA Middleware significant work remains to adequately assure interoperability of the elements of a large-scale LVC simulation activity.

Acknowledgments

The TENA Middleware was originally developed for the Foundation Initiative 2010 Project under contract 1435-04-01-CT-31085 in which the regulations from DFARS 252.227-7013 and 252.227-7014 are enforced. This paper describes the work of a team of software professionals, including the author and the current members of the TENA Development team: Steve Bachinsky, Horacio Blanco, Matt Carter, Mike Fogus, Marvin Greenberg, Rob Knight, Drew Lewis, Braden McDaniel, Dave Meyer, Brian Miller, Sean Paus, and Joe Seward.

The entire TENA development team would like to recognize the key contributions of the greater TENA team, including George Rumford, Kevin AIX, Gene Hudgins, Kurt Lessmann, Jason Lucas, Ed Powell, Keith Poch, Dee Salazar-Beddard, and Jerry Santos. Most importantly, we would like to thank the range engineers and system developers for their collaboration in making TENA a useful capability of the DoD.

References