Assessment of Complex System of Systems in a Distributed Live/Virtual/Constructive Environment

Kenneth R. Allred, Jr.
C. Todd Sutherland

July 29, 2014

The DoD has considerable geographically distributed Live/Virtual/Constructive (LVC) resources that it maintains. Depending on the type of test and assessment, one or several of these LVC resources might be involved in testing a subcomponent of a weapon, a weapon system, or a full System of Systems (SoS). In order to maximize the capabilities of the resources, it is desirable to establish a LVC distributed environment (LVC-DE).

Distributed testing can encompass the areas of Live, Virtual, and Constructive and a combination of the three. Tests involve people, hardware, and software and are conducted over a variety of parameter sets. The logistics of moving personnel, test items, and support infrastructure adds considerable expense and risk to the process of tactical system testing. The capability to allow interactions between distributed LVC capabilities and any other participants and/or stakeholders through the use of existing Modeling and Simulation (M&S) capabilities data networks would provide an immediate cost savings and risk reduction for the test program.

Kenetics Incorporated has extensive experience in the strategy, planning, design, development, integration and sustainment of LVC-DE capabilities such as the US Air Force Integrated Collaborative Environment (AF-ICE), The US Army Modeling Architecture for Technology Research, and Experimentation (MATREX), and the Joint US Air Force (Lead), Army, and Navy Net-Centric Weapons Test and Evaluation Environment (NCWTEE).

Based on our experience and lessons learned, we provide the following to explore and develop a set of tenets for an ideal LVC-DE architecture that would support use across multiple boundaries (e.g. performance assessment and training). It does not develop an architecture but rather provides the tenets, architecture approaches, and trade-offs associated with problems with specific domain types. We conclude with an assessment of the tenets and architecture approaches and suggest an approach utilizing open systems frameworks to enable the development of highly available, composable, secure, LVC distributed systems for mission critical applications supporting performance assessment and training applications across the SoS lifecycle.
Works Cited


LVC-DE Operational Context (LVC-DE OC)
The Joint Staff (JCS) developed, and the JROC endorsed the Joint Operations Concepts document (JOpsC) as part of a capabilities-based analytical construct that supports the Joint Capabilities Integration and Development System (JCIDS) and Joint Requirements Oversight Council (JROC) decision making. The JOpsC document describes the operational concepts for a Joint Mission, and describes subordinate concepts such as the Joint Operating Concepts (JOCs) (Strategic Mission/Mission Thread), Joint Functional Concepts (JFCs) (Mission Tasks), and Enabling Concepts (Mission Capabilities).

The LVC-DE Operational Context (LVC-DE OC) should serve as a point of reference and discussion of the operational, functional, and systems within a Joint Mission in support the JOpsC and the JOpsC subordinate documents (e.g. JFC Description Documents via JCIDS). The purpose of the LVC-DE OC is to provide the starting point for JCIDS analysis (FAA, FNA, FSA, etc.) and the envelopment of the JCIDS embodied architectures in regards to constructing a LVC-DE.

The LVC-DE OC defines the mission environment in which the subsystem/system/SoS is designed to operate, and provides the basis for developing a domain specific LVC-DE. The LVC-DE replicates, to the extent possible, the “real-world” mission environment in a repeatable test environment. Depending upon the complexity of the capability and system/SoS to be tested, the LVC-DE may consist of numerous, geographically separated entities and employ any combination of live, virtual, and/or constructive “participants.” The mission statement and end states to be achieved during test, and as well as mission objectives, blue forces, threat, and environmental aspects are critical in defining the LVC-DE OC, and provide a foundation for determining the scope or intended use of the LVC-DE. A properly developed LVC-DE OC defines the operational context, overall structure, major elements, and objectives to focus implementation of a LVC-DE. This unified LVC-DE OC is then used in follow-on development of the test scenario, vignettes, and trials, as well as acting as the referent for verification and validation.

The main advantage and disadvantage of network-centric SoS is the capability to field system-of-systems (SoS) providing an integrated capability. The advantage is that a single system or subsystem can be tested to see if it meets the design specifications. The disadvantage is that when that system or subsystem is combined with other systems and subsystems, the assessment must then address the performance of the new system and how well the individual components work within the aggregate system. The complexity of modern weaponry will entail more SoS testing and therefore strengthening the requirements for a LVC-DE.

One of the main benefits that can be derived from an LVC-DE is to overcome the sample size problem where the LVC-DE fully supports and validates flight testing. Major decisions on moving a new weapon into the realm of a full operational capability are generally made on very small sample sizes due to limitations on the M&S framework and simulation based Acquisition (SBA) practices. The statistical reliability of the data for small samples is always problematic. If the sample sizes are increased through the use of an LVC-DE capability, the statistical significance of the tests will be improved.

The basic prerequisite for use of an LVC-DE is having sufficient data upon which to build, verify, validate, and accredit (VV&A) the models along with the execution in a distributed environment. The M&S components and integrated M&S components that reside within the LVC-DE must stand as a principal source of confidence for BMDs performance over the entire range of operating conditions. To collect sufficient data on which to base the VV&A strategy we must increase the number of objects tested (ideal) or the number of data points collected. Since the probability of getting more flight tests is slim, the focus must be on increasing the amount of data when these flight tests occur, take advantage of hardware-in-the-loop (HWIL) capabilities (model-test-model), and extending common data collection and analysis across the full life-cycle of the system or system-of-systems.
The Question of Interoperability

Interoperability issues must be addressed in order to achieve the goal of network-centric testing. There are a number of issues that must be addressed. A major factor that must be considered is the integration of OEM and legacy capabilities that may fall outside of DOD control. Another factor is the ability of the DOD to provide the common environment (e.g. common truth services, enforcement of standard interface definitions, infrastructure, etc.) in which to rapidly and effectively integrate OEM and legacy capabilities into a validated SoS test capability.

If the assumption is made that the goal is to improve testing and to achieve the benefits of a network-centric design in the LVC-DE, then interoperability issues could be addressed in the following four (4) ways:

1. Common open-systems architecture
2. Reusable M&S components
3. Processes and standards for developing and distributing reusable M&S components,
4. Standard interfaces for development of LVC-DE capabilities and range instrumentation (data exchange, networks, software components, exposed external interfaces).

In order to address the interoperability issues and explore the benefits of the four (4) ways mentioned above, an LVC-DE Enterprise Architecture (LVC-DE EA) must be developed and used as the baseline for the overall acquisition strategy of an evolving LVC-DE implementation.

LVC-DE Enterprise Architecture Tenets

The development of a LVC-DE architecture requires the acceptance of certain concepts or premises which are accepted as core tenets that must be preserved. From those tenets, a Reference Model for the LVC-DE can be developed which describes overarching factors that the architecture should embody. The following sections will describe what we consider to be the tenets by which an architecture for and LVC-DE should be developed in light of the LVC domain areas described above. In actual practice, complete adherence to these tenets may not be practical or reasonable, but exceptions should be documented with technical justification for variance.

Domain: In software engineering, a domain is a software area that contains systems sharing commonalities and variabilities. The definition of a domain does not answer detailed questions of scope, but clearly includes and excludes broad classes of systems. Assumptions of commonality and exclusion identify the common features of systems in the domain, thereby establishing a set of relationships. The definition of the domain therefore bounds the problem space through the focus of intended use.

Components: The object-oriented paradigm delineates the components of a domain into objects. As in the real world, component objects can contain other objects or be contained within other objects. The number of components depends on the granularity chosen by the domain architect. Components can be real or simulation models. Real objects can exist in any geographical location in the domain. Component models can exist on the same computer platform or on different platforms. All components within the domain communicate with one or more other components (A standalone computer program will at the very least communicate with some output device.) Components must have at least one exposed interface to handle connectors. The conduit between components is through a connector, and the connector is attached to the component through the exposed interface.

Connectors: The communication between components can be abstracted as a connector. Since connectors separate communication mechanisms from components, they increase the reusability of components (the same component can be used in a variety of environments with different communication primitives). Connectors have standardized interfaces that can connect to the complementary interface of a component.
Common Services and Service Interfaces: Common Services provide independent, integrated, and common truth for an LVC-DE. This approach has proven successful in many large scale and complex LVC-DE implementations. This allows maximum reuse and avoids conflict with representations made within each LVC-DE capability. For example, components in the LVC-DE can share electro-magnetic phenomenology, lethality, environment, threat, and communications effects services.

Neutrality: The two main obstacles to reuse are component-to-component communication and data interchange. The use of connectors provides communication neutrality. Data can also be neutral if it is self-describing and therefore transparent to the receiver.

Granularity: Granularity is the relative size, scale, level of detail, or depth of penetration that characterizes an object or activity. An application constructed with more finely granular objects (i.e. a lower number of functions per object) is likely to be more easily maintained because objects should be smaller and less complex. Granularity is determined by the focal level selected during the decomposition process.

Operational Focal Level: the plane (level) of attention within a hierarchical system. The number of components represented in any domain will increase or decrease depending on which level attention is focused and some method of aggregating or disaggregating the constituent elements. The basic element of an army is the soldier and as the focal level moves up through the hierarchical structure of command, the arbitrary aggregations of platoon, company, brigade, etc. make it easier to handle logistically rather than trying to consider hundreds of individual entities.

Composability: If the domain is described by the components and their interactions (communications), then it should be possible to assemble components into a domain representation. One definition of composability is the ability to rapidly assemble, initialize, test, and execute a system from members of a pool of reusable, interoperable software elements (e.g. Software components or applications). Composability might also be considered as a measure of effectiveness of how well a given set of components and connectors will/can work to form a particular system.

Coupling: The interconnections and interactions between components to represent a behavior are known as coupling. Coupling is achieved through interfaces. The higher the degree of coupling, the likelihood is that it will be objectively harder to deal with since the interconnections and interactions will be more complex. As the components become more decoupled (independent), the potential for reuse should increase since they should be easier to modify. The amount of coupling is dependent on several factors including Quality of Service and timing dependencies.

Measure of Coupling: Any composable system can be described by a measure of coupling (MOC) between the components. The MOC of a particular system is related to the timing dependency and the Quality of Service Box (QB) demands required by components using a specified connection. Within any given composable system the MOC will vary from decoupled (event driven with minimal QB) to tightly coupled (scheduled with high QB).

Scalability: Generally, scalability is considered as the capability of a system to maintain a desired performance level as the volume of work increases for a given amount of resources. Conversely, the system should be capable of maintaining efficiency when the volume of work decreases.

Quality of Service (QoS): The Quality of Service (QoS) property definition usually has three parts: (1) a specification of what the property should be, (2) predefined limits of that property, if any, and (3) the actual, current reading of that property. QoS is generally composed of multiple requirements.

Quality of Service Box (QB): In order to speak about QoS as a collection when taken together, Pu (2001) coined the term QoS Box or QB (cube). Extending the concept of collective properties further, components which interact and require comparable level of coupling can be considered a QBGroup.
Predictability: The capability to determine how a composable system will react for a given QB and the available resources. There is a measure of predictability between two events when there is causality, which is the extent one event is caused by the other.

Timing Dependency: The interaction of components is generally actuated by events, schedulers, or both. Event actuated components are mostly unpredictable as to the occurrence of an action while schedule actuated components operate based on a timing function. Event-based systems are generally real world components, whereas scheduled systems are generally comprised of models.

Component Interaction: Interaction between components in the system involves an exchange of data or the initiation of an action. Actions are considered to be command & control functions (e.g. A remote procedure call would be a command). Data are considered to be any action-less interaction, however the receiver might initiate an action on receipt of data (e.g. process the data into a database).

Data Transparency: The concept of data transparency provides that the sending component does not have to know a priori what format the receiver requires for data. Data incorporating the appropriate contextual metadata can be understood by the receiver since the type and value of the data can be determined on receipt. Data transparency is not necessarily the best means of data transfer in tightly coupled systems. However, the greater the data transparency, the greater the interoperability.

The tenet of data transparency is shown in Figure 3. When components, connectors, and transparent data are used together, the architecture becomes universally stable. This does not imply anything concerning optimality. Optimal solutions are optimal for the specific set of trade offs and are dependent on the definition of optimal within the problem space. It is possible however, to increase optimality across our entire problem domain by using domain specific compartmentalization.

Interface to the User: All components with externalized interfaces must respond to a minimal set of user commands. Certain LVC-DE setup, command and control functions are required for any system to function properly. A universal method of invoking administrative commands would best be accomplished through the use of a single method that accepts a transparent data input.

Emergence: A LVC-DE lends itself to the study of system of systems (SoS) interactions. One of the undesired qualities of SoS is the exhibition of emergent properties. Chalmers (1990) wrote that emergence can be defined in several ways:

1. High-level patterns and structure emerge from simple low-level rules.
2. Emergent high-level properties are interesting, non-obvious consequences of low-level properties.
3. Emergence is the phenomenon wherein complex, interesting high-level function is produced as a result of combining simple low-level mechanisms in simple ways.
4. Emergence is the phenomenon wherein a system is designed according to certain principles, but interesting properties arise that are not included in the goals of the designer.

Architectural Stability: (Bahsoon, 2003) describes architectural stability as "the extent an architecture is flexible to endure evolutionary changes in stakeholder requirements and the environment, while leaving the architecture intact. Stable software architecture adds to the software system and to the enterprise owing the architecture a value. The added value is attributed to flexibility and the options that flexibility creates over the evolutionary periods of the software system. The added value under the stability context is strategic in essence and not immediate. It takes the form of (i) accumulated savings through enduring the change without “breaking” the architecture (necessitating changes to the architectural structure, architectural topology, or even the underlying architectural infrastructure); (ii) supporting reuse; (iii) enhancing the opportunities for strategic “growth”; and (iv) giving the enterprise a competitive advantage by banking the stable architecture like any other capitalized asset."
Review of Past and Current Architecture Approaches

The following descriptions of various architectures have been culled from published sources. In some cases (where noted) the text comes from documents published by the organization or principle author of the original document. The foregoing assessments of the various architectures should be construed as neither an endorsement nor a criticism.

Review past and current thinking in private industry

SOA: The goal of a service-oriented architecture (SOA) is to create a loosely coupled IT infrastructure. A service-oriented architecture is essentially a collection of services. A service is a component, which can communicate with other components through some form of connection service. Therefore, components and connections are both services. CORBA is an example of a SOA in which the components (CORBA objects) communicate with each other through the ORB component, which acts as the connector.

The primary architectural point is that the interfaces to the services be decoupled from the implementation. This requires either developing a client with multiple interfaces or the use of an abstraction layer that handles the communication issues. This abstraction is called the Service-Oriented Interface (SOI) and is implemented through the use of a middleware such as a webserver or message server. Other architectures also use this middleware approach, for example HLA and TENA.

SOA Comment: SOA architectures are designed for optimal reuse. Through the use of data neutrality and component decoupling, SOA is the most flexible architectural style for enterprise use. However, SOA would not handle the requirements for real time operation or the development of complex simulations or compute intensive tasks.

Eisenberg asks in his paper on SOA (Eisenberg, 2004), “What's the chief constraint in trying to realize the SOA vision? Primarily, it's that current applications are tightly coupled while the underlying SOA technology is still evolving. Therefore, it may not be possible—at least not in a cost-effective manner—to achieve loosely coupled perfection given the existing environment of tightly coupled, business-critical applications”

Model Driven Architecture (MDA): [Source: Object Management Group MDA documentation] The Model Driven Architecture (MDA) is a new way of writing specifications and developing applications, based on a platform-independent model (PIM). A complete MDA specification consists of a definitive platform-independent base UML® model, plus one or more platform-specific models (PSM) and interface definition sets. Each describing how the base model is implemented on a different middleware platform. A complete MDA application consists of a definitive PIM, plus one or more PSMs and complete implementations, one on each platform that the application developer decides to support.

MDA Comment: MDA is an effort to make the development and implementation of an architecture a more robust and structured process. It adheres to many of the tenets and focuses on the implementation of the platform independent models.

Enterprise Architecture (EA): (The Federation of Enterprise Architecture Professional Organizations, 2013) "a well-defined practice for conducting enterprise analysis, design, planning, and implementation, using a holistic approach at all times, for the successful development and execution of strategy. Enterprise architecture applies architecture principles and practices to guide organizations through the business, information, process, and technology changes necessary to execute their strategies. These practices utilize the various aspects of an enterprise to identify, motivate, and achieve these changes.” Enterprise Architecture implementation is essential to Architecture Stability and all other aspects of acquiring and sustaining LVC-DE capabilities. If executed properly, Enterprise Architectures provide the syntactic and semantic interoperability between design models, code, and implementation throughout the LVC-DE capability life-cycle promoting effective configuration control and management.
Review past and current thinking in academia

Domain Specific Software Architectures (DSSA): (Czarnecki, 2014) a domain-specific software architecture (DSSA) consists of a domain model, reference requirements, and a reference architecture. A DSSA forms the basis for devising a domain-specific development environment (DSDE) which supports the configuration of reusable components.

Each domain has its own set of typical requirements that an application in this domain has to satisfy. A domain model provides the vocabulary to formulate the reference requirements. Reference requirements are differentiated into functional and non-functional requirements. The latter will constrain the architecture and the implementation. Most importantly, the required variabilities in the domain have to be explicitly specified. Rapid prototyping can be used to test an architecture against the reference requirements.

DSSA Comment: The DSSA as described by the author, follows the pattern of component and connector as do most of the modern architectures. This is close to what our intended architecture would be like. The use of explicitly defined variability is problematic in that it might preclude emergent behavior within the system depending on the implementation.

Review past and current thinking in government

Distributed Interactive Simulation (DIS): Because [in (Hardt, 1998)] UDP provides the necessary and desired interoperability, it also allows an important feature of distributed interactive simulation. This feature that drives the DIS network requirements is the ability to work with output to and input from humans across distributed simulators in real time. This feature places tight limits on latency between hosts and also means that any practical network will require multicasting to implement the required distribution of all data to all participating simulators. Large distributed simulation configurations are expected to group hosts on multicast groups based on sharing the same sensor inputs in the virtual environment. This can mean a need for hundreds of multicast groups where objects may move between groups in large numbers at high rates. The overall total data rate (the sum of all multicast groups) is bounded, but the required data rate in any particular group cannot be predicted, and may change quite rapidly during the simulation.

DIS Comment: One of the early attempts at large scale distributed simulation architectures. DIS was designed for simulation and is not well suited for problems such as daily data acquisition. The implications of the network demands place the QB on this architecture to be very high. The Department of Defense mandated its replacement by HLA in 2001.

High Level Architecture (HLA): [Defense Modeling and Simulation Office] The HLA is a general purpose architecture for simulation reuse and interoperability. The HLA was developed under the leadership of the Defense Modeling and Simulation Office (DMSO) to support reuse and interoperability across the large numbers of different types of simulations developed and maintained by the DoD. The HLA Baseline Definition was completed on August 21, 1996. It was approved by the Under Secretary of Defense for Acquisition and Technology (USD(A&T)) as the standard technical architecture for all DoD simulations on September 10, 1996. The HLA was adopted as the Facility for Distributed Simulation Systems 1.0 by the Object Management Group (OMG) in November 1998 and updated in 2001 to reflect the changes resulting from commercial standardization of the specification under the IEEE. The HLA was approved as an open standard through the Institute of Electrical and Electronic Engineers (IEEE) - IEEE Standard 1516 - in September 2000. In November 2000 the Services and Joint Staff signed the HLA Memorandum of Agreement identifying the HLA as the preferred architecture for simulation interoperability within the DoD.

HLA Comment: HLA is the defacto standard for distributed simulations within DoD. It has severe limitations when it is used to handle event driven, dynamic, and unpredictable environments due to the requirement for the call to “tick” which maintains the time synchronization aspects of the simulation.
Test and Training Enabling Architecture (TENA): [TENA-SDA] TENA consists of:
• An operational architecture view that includes a concept of operations for how to use TENA
• A technical architecture view detailing rules and standards,
• A domain-specific software architecture view that contains a common metamodel, a common object model, a common infrastructure, and a common technical process,
• An application architecture view focused on how to build an application,
• A product line that identifies many utilities, tools, and gateway applications necessary for TENA to operate.

The most important technical driving requirements are interoperability, reuse, and composability. The most important degree of interoperability for TENA is termed “semantic interoperability,” interoperability built upon the foundation consisting of a common language and context for communication. TENA builds upon the HLA, but provides additional services specifically tailored to LVC-DE implementation.

The TENA Domain-Specific Software Architecture (DSSA) explicitly addresses interoperability by specifying four fundamental parts of TENA: a common meta-model, a common object model, a common software infrastructure, and a common technical process. The TENA object model represents the common language and common interfaces shared by all TENA applications. The common software infrastructure supports TENA applications in their use of the TENA object model throughout the logical range process. Finally, the common technical process provides guidance on how to create logical ranges and TENA-compliant software consistent with the overall TENA ConOps.

TENA Comment: As a DSSA variant, TENA has the same strengths and weaknesses. The implications of interoperability via the four fundamental parts may preclude extensibility and interoperability, especially in the areas of information exchange.

XMSF: (Moves Institute, 2014) The Extensible Modeling and Simulation Framework is defined as a composable set of standards, profiles and recommended practices for web-based modeling & simulation (M&S). XML-based markup languages, Internet technologies and Web Services will enable a new generation of distributed M&S applications to emerge, develop and interoperate. Many of the new challenges will include supporting existing technologies in a loosely coupled, distributed environment, which does not lend itself easily to capabilities we have come to expect in standalone and/or tightly coupled, distributed environments such as time management.

The precepts of XMSF are:
• Web-based technologies applied within an extensible framework will enable a new generation of modeling & simulation (M&S) applications to emerge, develop and interoperate.
• Support for operational tactical systems is a missing but essential requirement for defense M&S application frameworks.
• An extensible framework of XML-based languages can provide a bridge between diverse M&S requirements and open/commercial web standards, while continuing to support existing M&S technologies. The logical implication of data being machine-readable is that the data representation will need to be structured and self-defining.
• Compatible and complementary technical approaches are now possible for model definition, simulation execution, network-based education and training, network scalability, and distributed animation of 2D/3D graphics presentations.
• Web-based approaches for technology, software tools, content production and broad usage provide best business cases from an enterprise-wide (i.e. worldwide) perspective.

XMSF Comment: The essence of XMSF is akin to the SOA concepts but extending those ideas to include modeling and simulation. The XMSF use of data transparency is a significant factor.
**DoD Architecture Framework (DoDAF)**: The Department of Defense Architecture Framework (DoDAF), Version 2.0 is the overarching, comprehensive framework and conceptual model enabling the development of architectures to facilitate the ability of Department of Defense (DoD) managers at all levels to make key decisions more effectively through organized information sharing across the Department, Joint Capability Areas (JCAs), Mission, Component, and Program boundaries. The DoDAF serves as one of the principal pillars supporting the DoD Chief Information Officer (CIO) in his responsibilities for development and maintenance of architectures required under the Clinger-Cohen Act. DoDAF is prescribed for the use and development of Architectural Descriptions in the Department. It also provides extensive guidance on the development of architectures supporting the adoption and execution of Net-centric services within the Department.

**LVC-DE Enterprise Architecture Assessment**

Our target architecture must address the requirements of distributed testing at the varying scales and complexities as outlined in the problem space discussion. In order to gain a better understanding of what this might entail, let's briefly outline a set of broad LVC-DE implementations for some hypothetical domains (e.g. R&D, DT/OT).

Looking at the entries in Table 1, it is apparent that we would have a problem with using a single implementation to address all of the possible uses of the architecture even while attempting to address only a limited number of the operational tenets. The choice of any particular implementation of the architecture would necessitate accepting trade offs relevant to a prioritizing of the tenets. For example, if the primary goal were execution speed, then tailoring the software to specific hardware platforms and data structures would be the trade off over scalability and neutrality. Is it possible to have an architecture that can adhere to the tenets, meet all implementation requirements, and not suffer from instability?

<table>
<thead>
<tr>
<th>Timing</th>
<th>QB</th>
<th>Data Format</th>
<th>Execution Speed</th>
<th>External Dependencies</th>
</tr>
</thead>
<tbody>
<tr>
<td>Domain 1</td>
<td>Event</td>
<td>Low</td>
<td>Variable</td>
<td>Low</td>
</tr>
<tr>
<td>Domain 2</td>
<td>Mixed</td>
<td>Moderate</td>
<td>Variable</td>
<td>Moderate</td>
</tr>
<tr>
<td>Domain 3</td>
<td>Scheduled</td>
<td>High</td>
<td>Structured</td>
<td>High</td>
</tr>
<tr>
<td>Domain 4</td>
<td>Mixed</td>
<td>Variable</td>
<td>Variable</td>
<td>Variable</td>
</tr>
</tbody>
</table>

Consider the listing in Table 1, and the brief examination of some representative architectures. There is little likelihood that a single optimal solution exists that can cover all four of the hypothetical domains. That is, no single architecture can handle all problem domains equally well. However, if we consider each of the four as a domain specific compartmentalization of the problem space each can be implemented with as much optimality as possible. Each compartment would be considered an aggregate component and could communicate with the other aggregate components through the connectors. The LVC-DE must be developed in an architecturally stable manner to allow:

- interoperability across disparate domains;
- tunable performance (e.g. scale)
- flexible domain implementation across test and training boundaries.
Table 2 describes how each of the overarching LVC-DE operational requirements is met with the architecture tenets.

Table 2: LVC-DE Operational Requirements Mapped to Architecture Tenets

<table>
<thead>
<tr>
<th>LVC-DE Operational Requirement</th>
<th>Interoperability Across Disparate Domains</th>
<th>Tunable Performance</th>
<th>Domain Flexibility across test and training boundaries</th>
</tr>
</thead>
<tbody>
<tr>
<td>LVC-DE Tenets</td>
<td>Common / Composable User Interfaces</td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>• neutrality</td>
<td>• granularity</td>
<td>• components</td>
</tr>
<tr>
<td></td>
<td>• component interaction</td>
<td>• operational focal level</td>
<td>• connectors</td>
</tr>
<tr>
<td></td>
<td>• data transparency</td>
<td>• composability</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• coupling</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• scalability</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Quality of Service</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• predictability</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• timing dependency</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Measure of Coupling</td>
<td></td>
</tr>
</tbody>
</table>

The only tenet that we have not included in the above categorization is emergence. Since the property of emergence is exhibited through the interactions of independent or quasi-independent components, this tenet would be present primarily in loosely coupled architectures such as SOA or XMSF.

Let’s return to the question previously asked. Is it possible to have an architecture that can adhere to the tenets, meet all real-time and non-real-time implementation requirements, and not suffer from instability? One approach that can answer this question is to use (1) EA to provide the necessary definition of structure and behavior; (2) some of the concepts of DSSA to provide the necessary flexibility to ensure implementation of a stable architecture; and (3) some of the concepts of SOA to provide component modularity and reuse.

The recognition that the architecture must be flexible to domain specific implementation places an LVC-DE in the category of an enterprise architecture. Under the LVC-DE Enterprise Architecture (LVC-DE EA) would be DSSA implementations that adhere to the LVC-DE operational requirements and tenets. Maintaining architectural stability under this structure will require some constraints on any DSSA implementation.

The primary requirements for any DSSA will be the implementation of the component/connector paradigm and data transparency. Components are nothing more than objects, which must adhere to a uniform interface standard allowing for the interchangeability of connectors. Once the interface specification is established, components can be developed without regard to the transport mechanism that will be used at run time. If the composed system can be loosely or fully decoupled, the connector used might provide a message-oriented middleware. If the application is a complex distributed simulation, the connector might provide a more appropriate connection type.

The question might be raised that in some circumstances the DSSA might have to violate the interoperability and domain flexibility requirements in order to provide an adequate level of performance. This is not a problem when components are not seen as only “atomic.” An atomic component is one that has only a single function and does not incorporate any other components. An aggregate component is one that is made up of two or more components. From the viewpoint of a LVC-DE EA, only the aggregate component must adhere to the component/connector interface requirements. As long as the internal structure of the aggregate does not have an exposed interface, the aggregate will work in the system in the same manner as any other component. This is simply the principle of encapsulation used in
object oriented programming. Of course for maximum reuse, fully interchangeable components of the aggregate must adhere to the LVC-DE EA.

The maximum interoperability within the LVC-DE EA comes through the addition of data transparency. Systems and System of Systems tend to have data collected over the life-cycle that are variable in content and structure when examined from the end user point of view. This gives the impression that creating a structured data design would be problematic at best. However, the data may actually have commonality when the appropriate focal level is applied. For example, channel number can identify data collected through the use of data recorders. The desired goal then, is to understand the meaning of the data within the context of a given test at a specific time. There is considerable research effort being applied to the use of XML as the means of providing data transparency through the application of contextual frames of reference. The XMSF project aims to achieve data transparency (interoperability) through XML. The major drawback from a performance standpoint is the size of the XML document. Sosnoski recently published an article (Sosnoski, 2014) on improving XML transport performance using a variety of techniques. Sosnoski indicates that the expanding use of XML for data interchange has prompted considerable interest in developing ways to increase document processing speeds as well as decrease bandwidth requirements. Although XML may not be the perfect answer to the data transparency requirement at this point in time, it can already be used in some situations.

The recent efforts in developing SOA style implementations have also provided a mechanism for compositability. When components are established as services, they can be controlled through service interfaces. Common Services provide independent, integrated, and common truth for an LVC-DE. This approach has proven successful in many large scale and complex LVC-DE implementations. This allows maximum reuse and avoids conflict with representations made within each LVC-DE capability. For example, components in the LVC-DE can share common electro-magnetic phenomenology, lethality, environment, threat, and communications effects services.

Many agencies and departments are ideally suited for extension to an LVC-DE. These organizations have ranges and lab facilities that are widely dispersed geographically (e.g. Distributed nodes). The ability to federate models through an LVC-DE offers the potential for system engineering and performance verification that make full use of investments made in high fidelity and HWIL element representations.

Large organizations and joint operations are complex, geographically distributed, and cross multiple domains. It is understood that many practical difficulties of integrating and orchestrating an LVC-DE exist that mirror difficulties of integrating tactical systems in the real world. Often times, the EA design is limited by many monolithic legacy or OEM capabilities and their inability to support extraction of behavior representations of individual components. This is a universal issue with the design and implementation of most any kind of enterprise capability. Emphasis on architecture stability and enforcement of data transparency within the LVC-DE recognizes that LVC-DE components already exist although they may not have been designed and developed to operate together or in a geographically distributed manner. Emphasis should be placed on the reuse of the core engineering representations within these legacy elements interacting through provided common defined interfaces which adhere as much as possible to the domain requirements and tenets described above. Likewise, new components will be designed and developed with LVC-DE EA in mind.