MEASURES OF EFFECTIVENESS FOR LIVE, VIRTUAL, CONSTRUCTIVE INTEGRATED ARCHITECTURES

Gregory V. Funaro
Booz Allen Hamilton, Inc.
Suite 200, 6703 Odyssey Drive
Huntsville, AL 35806
256-922-6382
funaro_gregory@bah.com

Keywords: Architectural Patterns, Structural Complexity, Measures of Effectiveness, Sociotechnical

ACKNOWLEDGEMENT: Booz Allen and the author would like to acknowledge and thank the Air Force Agency for Modeling and Simulation (AFAMS) for its support, guidance and information.

ABSTRACT: The Department of Defense (DoD) strives to improve Live, Virtual, and Constructive Integrated Architectures (LVCIAs) with the objective to assemble models and simulations (M&S) that create representations of a credible operating environment. Changes in the operating environment and the needs of the warfighter require a continued focus on developing corporate and cross-cutting business practices which improve visibility; accessibility; commonality; reuse and interoperability of M&S tools; and data and services. A robust M&S capability enables the Department to meet operational and support objectives across the diverse activities of the services, combatant commands, and agencies more effectively. The ability to determine the success of the strategic vision for DoD M&S depends on the ability to identify those measures that capture the attributes that empower DoD with M&S capabilities which effectively and efficiently support the full spectrum of the Department’s activities and operations.

This paper examines the current gaps in an M&S LVCI A and what is required to enable an effective LVCI A. It does so by examining the interaction between people, processes and technologies in the DoD M&S community. The needs and objectives of the LVCI A are readily available and there are many successes using LVCI A in training and testing. It has not been as successful in acquisition, analysis, experimentation and other communities. By examining architectural software patterns, software paradigms and ongoing interoperability standards, this paper identifies those measures that provide a better understanding of the M&S enablers. These measures of effectiveness will guide the M&S communities to satisfy the current needs of the LVCI A community.

1 Introduction

For over a decade, the ultimate goal of the DoD in M&S has been to create a LVCI A to quickly assemble models and simulations, as well as to create representations of a credible operating environment. A robust M&S capability enables the Department to more effectively meet operational and support objectives across the diverse activities of the services and agencies. Although LVCI A has been successful in a warfighter training capacity, LVCI A falls short in many aspects. In 1996, the Study on the Effectiveness of Modeling and Simulation in the Acquisition Process identified three groups where the LVCI A falls short: technical, managerial and cultural. [1] In 2007, the Modeling and Simulation Strategic Plan Development Summary Report discussed the need to take into account technological, management issues and
procedural realities. After ten years, the shortfalls are the same.

To better achieve the DoD’s M&S goal, this paper attempts to correlate LVCIA shortfalls to the technological capabilities required to enable both social and business needs. Socio-technical factors are explored in order to gain a better understanding of failing issues and identify metrics that determine the ability to empower the M&S community to both effectively and efficiently support the full spectrum of the Department’s activities and operations.

2 Approach

This paper will begin with a discussion on architectural patterns and their role in identifying technical and social solutions for LVCIA shortfalls, gaps and issues. Architectural patterns are sets of software design patterns with established concepts which solve specific problems and provide a rationale for the solution. It does not describe the implementation of the architecture nor the approach taken.

The architectural section is followed by a table of LVCIA gaps and solutions. The solutions are cast onto the enabling architecture pattern for each of the gaps.

Finally, a discussion on useful metrics which determine measures of effectiveness is presented. It will be shown that these measures can be used to determine the benefit and cost of a solution. Specifically, a discussion on structural complexity and goal-driven functional size is presented as a means for determining effort.

3 Background

The success of socio-technical systems is dependent upon end users participating in the system design process. An architectural description facilitates communication between end users and developers regarding the structure of the system. Four techniques are available for performing architectural descriptions: formal architecture description language (ADL); ANSI/IEEE 1471-2000 views or viewpoints adopted in 2007 by ISO; frameworks such as the DODAF and Zachman framework; and architectural patterns. Christopher Alexander first developed the idea of architectural patterns for buildings circa 1970. His idea influenced computer scientists to begin identifying patterns in the 1980s. The idea of patterns as applied to software systems first emerged in 1994 with Design Patterns: Elements of Reusable Object-Oriented Software.

In Architectural Patterns Revisited – A Pattern Language, Avgeriou and Zdun make a distinction between architectural patterns and architectural styles and describes a common ground set of architectural patterns. Architectural patterns focus on the problem-solution pair that occurs in a given context or architectural element, whereas architectural styles focus on components, connectors, control and data flow. This paper expands on the Avgeriou and Zdun approach by including the crosscutting design patterns as described in Patterns of Aspect-Oriented Design.

Cross-cutting concerns are defined within the domain of aspect oriented design patterns. Aspect oriented design is a new technology, which began gaining momentum around 2005. This type of design provides the ability to weave or compose aspects and modularize concerns. This ability offers the opportunity to reason through concerns that cut across a system, i.e. cross-cutting concerns. Aspect-oriented design is the technology that directly supports the need for an LVCIA to address cross-cutting concerns; a technology that improves system cohesion.

Most are familiar with object oriented development methods. These provide the ability to encapsulate or hide data behind a defined interface to support change with the Adaptor design pattern. The intent of object oriented design is to reduce system complexity by limiting the interdependency between components and decoupling the modules. One of the main goals of the object oriented paradigm is reuse.

But if the intent of object oriented design is to foster reuse, it raises the question as to whether the inability to effectively reuse models is due to a technological limitation, a cultural misunderstanding, a lacking policy, or a gap in the process. To answer this question consider Paul Graham’s essay The Hundred Year Language. Graham made the significant note that “somehow the idea of reusability got attached to object-oriented programming in the 1980s, and no amount of evidence to the contrary seems to be able to shake it free.”

Graham makes two points:
Object-oriented programming is reusable because it supports the software design process, as opposed to the component or object aspect.

Software libraries are good examples of reusability because they support a language, as opposed to because they are objects.

Graham further elaborates this statement by emphasizing that the benefit of object-oriented programming is not a technological solution. He says:

“I don’t think it has much to offer good programmers, except in certain specialized domains, it is irresistible to large organizations. Object-oriented programming offers a sustainable way to write spaghetti code.” [4]

Both the 1996 and 2007 reports identify motivation, advocacy, and governance as key enablers for a successful LVCIA. The questions that arise are the same for any quality attribute. What motive, advocacy or governance will ensure reusability? What is required of the LVCIA to support these enablers? What architectural elements are required to support both the end user and developer achieve a quality attribute?

4 Architecture

This section describes the architectural design patterns used to solve common problems and issues. Many of the architectural layers are well-understood and common architectural layers, while a few additional layers have emerged as the issues were examined. There are seven architectural layers in total and they are examined in pieces. Some of the architectures are technical and some are social.

The first two architectural layers follow Microsoft’s Integration Patterns [5]. These consist of an integration layer and a topological layer. Both layers contain supporting sub-layers as identified by standard design patterns. The integration layer focuses on data and services and represents the semantic and syntactic connectivity for interoperability. The topology layer focuses on network connectivity and the services joining distributed entities together. It focuses on location, network structure and channels. These two layers are technology enablers.

- Integration Layer
  - Entity Aggregation
  - Process Integration
    - Data Integration
      - Shared Database
      - Replication
      - File Transfers
    - Function Integration
      - Distributed Objects
      - Service Oriented
      - Message Oriented Middleware
    - Presentation
  - Topology Layer
    - Portal Integration
    - Point to Point
    - Message Bus
    - Broker
      - Direct
      - Indirect
    - Publish/Subscribe
      - List Based
      - Broadcast Based
      - Content Based

A representation layer provides a place in which to place models and support reusability. It provides support for the conceptual, implementation, experimental, simulated and federated models recommended by the National Research Council in Defense Modeling, Simulation, and Analysis. [6] The representation layer is a technology enabler.

- Representation Layer
  - Experimental Frames
  - Conceptual Models
  - Implemented Models
  - Simulators
  - Federates

An aspect oriented layer provides the end user access to the LVCIA through the supporting design patterns. The Aspect Layer follows the architectural patterns described by Noble, J., et al in Patterns of Aspect-Oriented Design. The aspect oriented layer is predominantly a social enabler, because of the design patterns that provide user access into an architecture and its cross-cutting capacity to accommodate users from the DoD communities, services and agencies.
Aspect Oriented Layer
- Spectator (Blackboard) – provides the capacity to monitor or trace the
- Regulator – regulates the behavior (Exceptions, method execution, abort, data transactions).
- Repair Person – provides the capacity to extend the functionality of legacy code.
- Role Based Access and Configuration
- Developer
- Manager
- Adaptor – provides the capacity to adapt to changes, i.e. evolve
  - Microkernel
  - Reflection
  - Interception
- Language Extension – provides the capacity to offer an abstract layer to the computational infrastructure.
  - Interpreter
  - Virtualization
  - Rule Based Systems

Booz Allen Hamilton is currently conducting a study for the Air Force Agency for Modeling and Simulation (AFAMS) to assist in the development of a more effective LVCIA. The following table categorizes and summarizes all the gaps within this study and identifies proposed solutions through the help of subject matter experts.

<table>
<thead>
<tr>
<th>Gap</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>Interoperability</td>
<td>Communication</td>
</tr>
<tr>
<td></td>
<td>Semantic &amp; Syntactic Integration</td>
</tr>
<tr>
<td></td>
<td>Configurable Gateways</td>
</tr>
<tr>
<td></td>
<td>Common Representation</td>
</tr>
<tr>
<td></td>
<td>Programmatic Authority</td>
</tr>
<tr>
<td>Composition</td>
<td>Standards &amp; Policy</td>
</tr>
<tr>
<td></td>
<td>Data Strategy</td>
</tr>
<tr>
<td></td>
<td>Service Agreement</td>
</tr>
<tr>
<td></td>
<td>Catalog of Models and Ranges</td>
</tr>
<tr>
<td></td>
<td>Composable Models</td>
</tr>
<tr>
<td></td>
<td>Available to Users</td>
</tr>
<tr>
<td>Resource and Data Discovery</td>
<td>Data, Services, Ranges &amp; Representations visible to users</td>
</tr>
<tr>
<td></td>
<td>Discovery Tools</td>
</tr>
<tr>
<td>Data Mediation</td>
<td>Standards and Policy</td>
</tr>
<tr>
<td></td>
<td>Data Ownership</td>
</tr>
<tr>
<td></td>
<td>Service Agreements</td>
</tr>
<tr>
<td></td>
<td>Protocol Conversions</td>
</tr>
<tr>
<td></td>
<td>Base Object Models</td>
</tr>
<tr>
<td></td>
<td>Available &amp; discoverable to users</td>
</tr>
<tr>
<td></td>
<td>Data Conversion</td>
</tr>
<tr>
<td></td>
<td>Resolution Filters &amp; Service</td>
</tr>
<tr>
<td>Infrastructure</td>
<td>Tools for monitoring performance</td>
</tr>
<tr>
<td></td>
<td>Access to data, services and ranges</td>
</tr>
<tr>
<td></td>
<td>Time Management</td>
</tr>
<tr>
<td>Governance</td>
<td>Standards &amp; Policy</td>
</tr>
<tr>
<td></td>
<td>Cross cutting Tools</td>
</tr>
<tr>
<td></td>
<td>Behavior Regulation</td>
</tr>
<tr>
<td></td>
<td>Central Authority</td>
</tr>
<tr>
<td>Pre &amp; Post Event (planning)</td>
<td>Planning and Discovery Tools</td>
</tr>
<tr>
<td></td>
<td>Tools across data, services, ranges and models</td>
</tr>
<tr>
<td></td>
<td>Communities of Practice</td>
</tr>
<tr>
<td></td>
<td>Data Ownership</td>
</tr>
<tr>
<td></td>
<td>Model Ownership</td>
</tr>
<tr>
<td>Efficient Use</td>
<td>Available &amp; Discoverable to users</td>
</tr>
<tr>
<td></td>
<td>Expandable</td>
</tr>
<tr>
<td></td>
<td>Contract Programming</td>
</tr>
<tr>
<td></td>
<td>VV&amp;A Ownership</td>
</tr>
<tr>
<td>Requirements</td>
<td>Communities of Practice</td>
</tr>
<tr>
<td></td>
<td>Data Ownership</td>
</tr>
<tr>
<td></td>
<td>Model Ownership</td>
</tr>
<tr>
<td></td>
<td>Requirement Traceability Tools</td>
</tr>
<tr>
<td></td>
<td>Expandable</td>
</tr>
<tr>
<td></td>
<td>Contract Programming</td>
</tr>
<tr>
<td></td>
<td>VV&amp;A</td>
</tr>
</tbody>
</table>

The enterprise layer provides support for social networking and user collaboration. It is a social enabler.

Enterprise Layer
- Collaboration
- Social Network

Two additional layers that address issues and gaps are the verification, validation, & accreditation (VV&A) layer, for VV&A issues and a time management layer for event based and real time stepping. Time management is a predominantly a technical enabler and VV&A is predominantly a social enabler.

Verification, Validation and Accreditation

Time Management
- Real Time
- Discrete

5 Issues and Gaps

Each of the DoD services have recently been directed by the DoD to improve the LVCIA environment. The Air Force has been collecting issues and gaps from its user communities; acquisition; analysis; experimentation; education; planning; testing and training. [7] [8] [9] [10]
<table>
<thead>
<tr>
<th>Gap</th>
<th>Solution</th>
</tr>
</thead>
<tbody>
<tr>
<td>VV&amp;A</td>
<td>Data Collection Models Available to Users Ownership</td>
</tr>
<tr>
<td>Reuse</td>
<td>Social Network &amp; Collaboration Data &amp; Services understandable to user community Available to user community Cross Domain Solutions (Security)</td>
</tr>
<tr>
<td>Standards</td>
<td>Social Network Contract Programming</td>
</tr>
<tr>
<td>Security</td>
<td>Enterprise Data and Service Authority Configurable Topologies Data Filtering Cross Domain Solution DIACAP Tools</td>
</tr>
<tr>
<td>Workforce</td>
<td>Training Collaboration</td>
</tr>
<tr>
<td>Collaboration</td>
<td>Access to user community</td>
</tr>
<tr>
<td>Representation</td>
<td>Need for representative model Fidelity and Resolution Filters VV&amp;A</td>
</tr>
<tr>
<td>Common &amp; Consistent Interaction</td>
<td>Enhanced Integration Configurable Connections Security</td>
</tr>
<tr>
<td>Training Support</td>
<td>Organization Certifications Best Practice Standardize Data Capture, Playback &amp; Storage Data Visualization Tools</td>
</tr>
<tr>
<td>Fault Monitoring</td>
<td>Persistence</td>
</tr>
</tbody>
</table>

6 Architectural Solutions

The following table places the solutions and associated gaps onto an architectural framework which consists of the architectural layers and design patterns previously described. The table depicts the linear span of an issue with respect to a solution space. It also indicates which architecture, or architectures, enables each solution.

Gaps and issues are categorized into functional goals and are listed on the left hand side of the table. For example, the goal of interoperability and data mediation require an LVCIA with nearly all architectural patterns, obviously a large undertaking. The need for models, such as models for irregular warfare, behavior, common operating picture, weapons, etc is captured in the representation category. Though the need to create models to solve various DoD concerns is a large undertaking, it has little impact on an architectural design.

To assess the effort required in order to enable the LVCIA to support a solution, it is critical to find architectural metrics useful in determining the size of the solution, in addition to the complexity of its implementation. These metrics should be based around the needs of both the developer and end user.

The following discussions on effectiveness measures will utilize two recommended methods for determining architectural effectiveness: structural complexity and functional size. The example above shows representation to have a small functional size, and little dependency on other architectural patterns. Interoperability, on the other hand, has large functional size and is dependent on a multitude of architectural patterns, implying a large structural complexity.

Since the technical and social factors enable the success of the LVCIA, it is important to find effectiveness measures that include both social and technical domains. In Towards Measures for Software Architectures, [11] Chastek and Ferguson define architectural measures as follows: “Architectural measures are needed that directly indicate when a change is required in the software architecture, or that verify that the software architecture satisfies its goals.”
7 Measures of Structural Complexity

Structural complexity examines the organization of a system. Coupling and cohesion are common measures of structural complexity and are applicable to organizational structures, as well as software architectures. [12] Also, coupling and cohesion metrics are invariant to software paradigms, i.e. the programming language chosen to implement an architectural design pattern will not change the metric. Both coupling and cohesion play an important role in controlling structural complexity when designing, implementing and maintaining an enterprise, software or hardware architecture. The quality attributes related to cohesion and coupling are directly related to the functional goals of an LVCIA and are as follows:

- Maintainability
- Reusability
- Interoperability
- Flexibility
- Expandability
- Verifiability
- Survivability
- Portability

To gain further insight into how to achieve maintainability, reusability and other quality attributes, it is worth understanding the types of cohesion and coupling metrics. Studies indicate that cohesion and coupling,
although similar, do not contribute to the quality attributes in the same manner. Darcy, et al in *The Structural Complexity of Software: An Experimental Test* concluded that cohesion contributes to maintenance at a faster rate than decoupling. [12] In other words, it is easier to maintain and reuse models in an LVCIA with well defined functionality, i.e. high cohesion, than with architectures with a message based interfaces, i.e. low coupling. Together, a framework with well partitioned functionality and a message based interface optimizes the maintenance and reuse quality attributes.

In the definitions for cohesion, note how the architectural patterns naturally partition into functional groups, reducing structural complexity and enabling maintainability and other quality attributes. The functional separation of the layers then fosters sequential cohesion across the architecture.

The common types of cohesion from worst to best are as follows:

**Coincidental** - Parts of a module of frequently used functions are arbitrarily grouped; the parts have no significant relationship.

**Logical** - Parts of a module are grouped because they are logically categorized to perform the same function, even if they are different.

**Temporal** - Parts of a module are grouped by the order in which they are processed and the parts are processed at a particular time in program execution. A function which is called after catching an exception, and which closes open files, creates an error log and notifies the user.

**Procedural** - Parts of a module are grouped because they always follow a certain sequence of execution. It is a function which checks file permissions and then opens the file.

**Communicational** - Parts of a module are grouped because they operate on the same data (e.g. a module which operates on the same record of information).

**Sequential** – Parts of a module are grouped because the output from one part is the input to another part like an assembly line. It is a function which reads data from a file and processes the data.

**Functional** - Parts of a module are grouped because they contribute to a single well-defined task of the module.

Common types of coupling, from high to low are as follows:

**Content** – One module depends on the internal workings of another module.

**Common** – Two modules share the same global data.

**External** – Two modules share an externally exposed data format, communication protocol or device interface.

**Control** – One module controls the logic of another by passing its information on what to do.

**Stamp** – Modules share a composite data structure and use only part of it, including what may be a different part.

**Data** – Modules communicate by parameters, each one being either a single data item or a homogenous set of data items which does not contain any control element.

**Message** – Modules are not dependent on each other and use a public interface to exchange parameter-less messages or events.

8 Measures of Goal Driven Functional Size

Clark, et al in *Subject-Oriented Design* [13] identifies three problems to be aware of in software paradigm methods, especially in object oriented design practices which promote design considerations more than user’s goals and needs.

1) Implementations can become large and monolithic, reducing maintainability, comprehension and reusability.
2) Systems become too specialized and too tightly coupled creating a misalignment between requirements and models.
3) Requirements are either scattered, implemented across objects; or they become tangled, a single model contributes to one or more requirements.

In an architectural approach, the focus is based on partitioned functional cohesion that decouples requirements into appropriate design patterns. This functional breakdown allows for a deeper analysis of the
inflows and corresponding outflows that have the largest influence on LVCIA enablers. It allows for the influence of the instinctual issues that are identifiable and provides insight into those gaps that have yet to be identified.

Works Cited


Author Biography

Gregory Funaro is an associate at Booz Allen Hamilton. He has been professionally involved in systems engineering, software engineer, modeling and simulation for over 35 years.