

ARCHITECTURE CONCEPTS FOR SIMULATION-BASED ACQUISITION OF COMPLEX SYSTEMS

Dr. Bipin Chadha and John Welsh
Lockheed Martin Advanced Technology Laboratories
1 Federal Street, A&E Building
Camden, NJ 08102
bchadha, jwelsh@atl.lmco.com

KEYWORDS

Architecture, Federation, Enterprise, Complex Systems, Simulation-Based Acquisition

ABSTRACT

The ability to make good design decisions early is a significant driver for simulation-based acquisition to effectively lower life-cycle cost and cycle time. Building virtual prototypes, enabling one to analyze the impact of decisions, achieves effective simulation-based acquisition processes. Virtual prototypes need to support a comprehensive set of analyses that will be performed on the product; hence, all aspects of product data and behavior need to be represented. Building virtual prototypes of complex systems being designed by a multi-organizational team requires new architectural concepts and redesigned processes. Implementation of these new architectures is complex and leveraging commercial technologies is necessary to achieve feasible solutions. One must also carefully consider the state of the current commercial technologies and frameworks as well as the organizational and cultural aspects of organizations that use these systems. This paper describes key architectural principles that one must address for a cost-effective implementation. The paper then discusses key architectural concepts and trade-offs that are necessary to support virtual prototypes of complex systems.

INTRODUCTION

Lockheed Martin, government, and industry partners, and supply chain members are developing and manufacturing large, complex systems. Using simulation-based acquisition and design strategies to develop cost-effective virtual

prototypes of these systems presents enormous challenges. One will use virtual prototypes, intended to support a product's entire life-cycle, for multiple purposes, including system engineering during conceptual design and for warfighters during training. Therefore, virtual prototypes must capture all information related to a product's definition and must provide mechanisms to incorporate all aspects of a product's behavior. Product complexity forces organizations to share the design and manufacturing of these products; therefore, the virtual prototype must be shared. Detailed knowledge of a subsystem typically resides with the supplying organization, so it is critical that the supplier organization creates and manages the virtual prototype of that subsystem. Management of the virtual prototype's complexity is important to ensure that developments focus on areas of sufficient benefit; otherwise, the implementation cost of the virtual prototype could be unbounded. Well-designed architectures are key to managing complexity. The following key goals will ensure a cost-effective virtual prototype:

- Create a multi-domain product model that is open, extensible, integrated, and synchronized.
- Integrate equations and/or behaviors with the relevant product data, thereby enabling evaluation of functional performance and optimization of life-cycle costs.
- Create executable representations of the system with multi-level fidelity to support multi-level analysis.
- Execute simulations for analysis and demonstration.

MYTH OF COMMONALITY IN ARCHITECTURE

The trend in information technologies is toward common systems, tools, and processes. While this is a trend away from disjointed systems, it is unrealistic to assume there will ever be a single process or system that satisfies every need in a multi-business organization. Figure 1

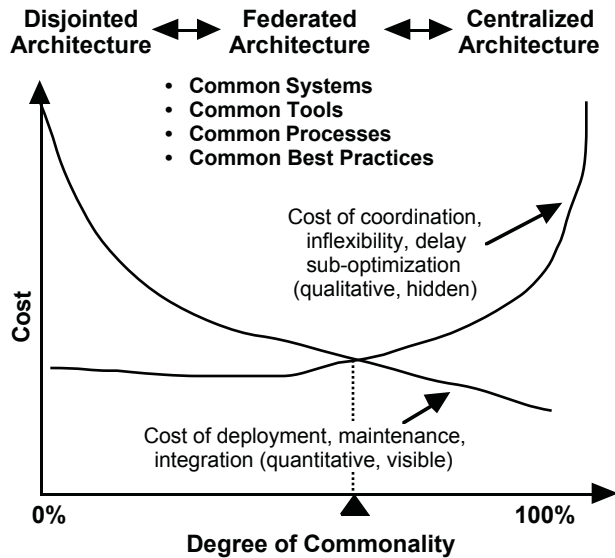


Figure 1. Federated Architecture Provides an Optimal Solution

shows diminishing returns as the degree of commonality increases beyond a limit. Figure 2 shows that hidden or ignored qualitative factors increase costs, while easily quantifiable and visible metrics decrease costs. In this example, maintenance, deployment, and integration represent quantifiable costs, while coordination and changes and implementation delays represent qualitative factors that cause significant cost penalties in overall solutions. Most

approaches ignore qualitative factors that become important as commonality approaches 100 percent. Existing architectures only account for technical aspects and ignore cultural and organizational aspects. The critical issue is not commonalities in architecture but understanding the trade-offs to determine the right amount of commonality for a given domain in a given context. The same can be said for any given standard. Standards should be evaluated in context to determine whether they are applicable, mature, and cost effective for the problem at hand. Applying a standard just because it is a standard is likely to produce less than desirable results.

Better virtual-prototyping architectures are needed to address this dichotomy among business needs and capabilities available with commercial off-the-shelf (COTS) solutions. A federated approach (Chadha 1997) provides the flexibility to deal with these current trend factors. Federation implies that one has established a degree of interoperability among business systems. Interoperability enables a shared-information area and provides a set of operations that can be initiated in a controlled fashion by one system operating on information in the shared/private area of another system. Members can join a federation to share information and functionality on a program and they can leave when the effort is finished. Federation provides a low-risk alternative to existing technology investment strategies, and it enables organizations to bring new tech-

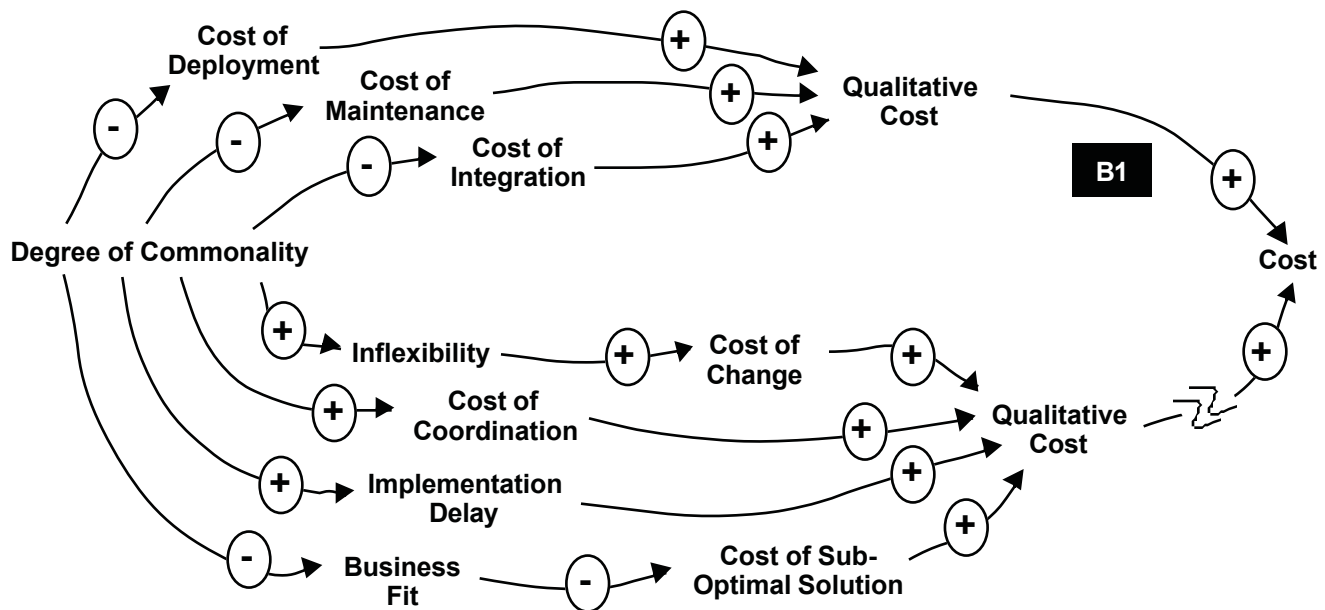


Figure 2. Architecture Trade-Off Model

nologies in a modular fashion instead of requiring a total upgrade of the internal business-system architecture (“*big bang*” approach). A federated architecture enables programs to implement a total-systems view and it also provides the ability to optimize supply chains at a global level.

GUIDING PRINCIPLES

Several principles guide the development of the architecture. These principles are useful in information technology and other domains, such as lean manufacturing (Womack, Jones, and Ross 1990) and systems thinking (Senge *et al.* 1999):

- Leverage COTS tools and technologies to the maximum extent possible.
- Focus on federated approaches instead of homogeneous information-system approaches.
- Provide information (and resources) only when needed.
- Provide only needed information (no more, no less).
- Do not carry information defects to the next step.
- Put in place a process that discourages the generation of defects.
- Information should be owned by the entity that is most suited to keep it current/accurate.
- Information must be accessible by those who have or may have a need.
- Account for soft/qualitative factors.

The focus is on *guiding principles* as opposed to hard-design constraints. This allows one to realize benefits in most cases and to avoid penalties associated with exceptional cases.

ARCHITECTURE FOR SIMULATION-BASED ACQUISITION

The principle objective of the virtual-prototyping architecture is to reduce the complexity of the system being designed to implement the virtual prototype. The architecture must also help achieve modularity and reuse of components that make up that architecture. Above all, the architecture must be able to evolve over time as the operating environment changes. This includes changes in requirements, expectations, organizational structures, business processes, and technologies. One must apply the principles of Cost as an Independent Variable (CAIV) analysis for systems design to the architecture for the virtual prototype.

Architecture Concepts

Rapidly evolving software, network, and associated technologies require innovative concepts to leverage their benefits in the architectural approach. This section provides an overview of some of these concepts or architectural patterns, which represent foundational technologies for virtual prototyping architectures.

Internet Architecture. The virtual-prototype architecture will need to be highly scalable and evolvable. The Internet provides an architecture that is highly distributed and scalable. It derives these properties via simple mechanisms, such as decentralized control, small set of ubiquitous standards, and hyperlinking information across independent web servers. It continues to evolve in a bottom-up fashion as content and new technologies are being added. Nodes on networks can be clients and/or servers, and each can perform the role that is necessary for operation. The ability of enterprise agents to hyperlink to each other across the network and to travel across clients and servers will provide essentially unlimited scalability for the architecture.

Common Object Request Broker Architecture. The Object Request Broker (ORB) is a basic architectural construct that sits between clients and makes requests. The ORB mechanisms can support all interactions among various architectural components.

One can statically define interfaces in an interface definition language (object management group IDL), which defines the types of objects according to operations that may be performed on them and the parameters to those operations. The IDL is language independent and bindings to several programming languages, such as C++ and Java, are available. Note that ORBs do not provide complete data-management functions or object-domain functionality. They facilitate the communication without necessarily understanding what is being communicated. They also interoperate via the Internet Inter-ORB Protocol (IIOP), which specifies a standardized interoperability protocol for the Internet.

Federated Architecture. A federation implies a loosely-coupled system distributed across the Internet or Intranet, where participants join or leave the federation without breaking it and where they function on their own when not a part of it (Figure 3).

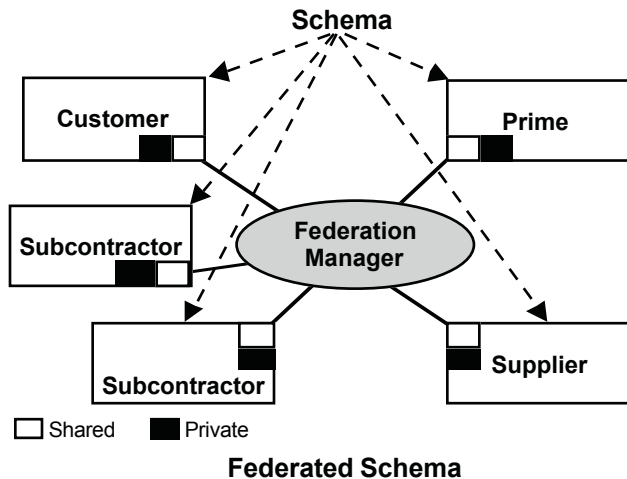


Figure 3. Federated Architecture Concept

The federation provides a low-risk alternative to existing technology-investment strategies, enabling organizations to bring new technologies in a modular fashion as opposed to the *big bang* approach, which requires replacement of most or all of the information systems for a technology upgrade. Mechanisms, such as smart-proxy objects, achieve federation.

Smart Proxy Objects. An innovative concept is the "proxy" object, which provides a surrogate or placeholder for another object to control access to it (Figure 4) (Gamma *et al.* 1994). A proxy object resides in the local environment and represents an object residing in a remote environment. The proxy handles requests made to it and forwards them to the real object for further processing, which returns the results back to the proxy object. The proxy then forwards the results to the requester. Therefore, clients only work with the proxy object in the local envi-

ronment. They are insulated from the details and how-to of dealing with objects in remote environments. These proxies are not the same as those in the ORBs; however, they will use proxy objects and services provided by the ORBs for their operation. Proxy objects provide tremendous flexibility in implementation approaches, enabling variable degrees of information replication and flexible behavior via tailorable business rules. Proxies can be implemented to various levels of complexity, starting from simple uniform resource locators to complex, tightly-coupled proxy objects.

Component-Based Architecture. A key concept that enables realistic and affordable development of the architecture is the ability to build in modular, reusable components. A component is a reusable software program that can be easily added to or combined with other software programs, enabling construction of sophisticated programs in a modular fashion. One can divide the architecture into components via well-defined interfaces. Interfaces allow small components to be grouped to create complex systems that would otherwise be impractical to build. Concepts of interfaces allow objects to be typed across many dimensions. Objects can adhere to many different interfaces without the burden of multiple inheritance. This also allows one to build information models from the bottom-up and not use the monolithic, top-down information model. The impact of component-based architectures is a significant gain in productivity. The use of this design strategy is rapidly gaining acceptance by the information-technology community. Both COM+ and Java use component-based architectures.

Systemic Behavior. Systemic approaches (Senge *et al.* 1999) differentiate between systemic behavior and compo-

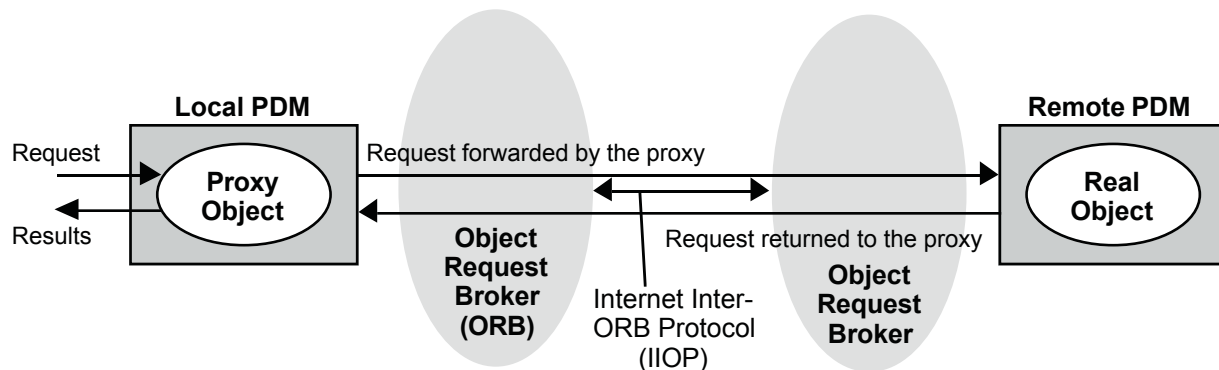


Figure 4. Proxy Object Concept

nent behavior. Systemic behavior is the behavior of a system that emerges due to interactions among components. This behavior is not a result of a collection of behaviors of the components. Complex dynamic systems exhibit a wide variety of systemic behaviors that can only be understood by analysis at the systemic level. The following characteristics of component interactions determine the systemic behavior:

- Everything is connected to everything.
- System exhibits properties and behaviors different from the parts.
- There are no independent variables.
- System structure drives system behavior (especially the feedback loops).
- Many effects are indirect and delayed.
- Many dependencies are non-linear.
- Behavior is dynamic.

Systemic issues also result in unpredictable behavior in complex systems leading to "system accidents" (Perrow 1984). Causes can be traced to tight coupling and complex interactions between components of the system. The virtual prototypes of complex systems therefore need to represent these interactions and couplings to be effective in predicting and avoiding some of these complex behavioral modes.

The systemic behavior is modeled via causal loop diagrams (Senge *et al.* 1994) as shown in Figure 5. Causal loops embed the properties of causes and effects, feedback loops, delays, and non-linearity among components of a system. Understanding the causal links enables the architects to decompose a system such that the complex interdependencies reside within a subsystem while few interdependencies remain between subsystems (Alexander 1964).

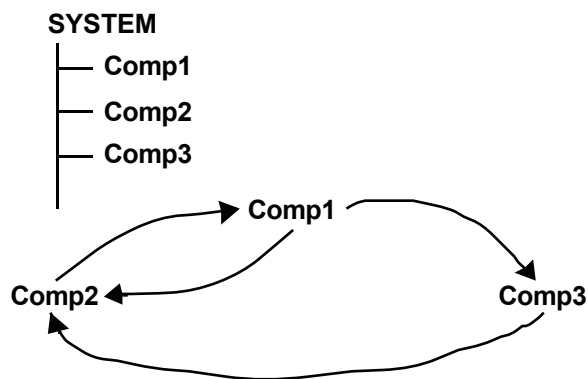


Figure 5. System Representation via Hierarchy and Causal Loops

Many systems engineering approaches either ignore these effects or model them simply as static interfaces.

Existing simulation approaches are attempting to address these issues by integrating various analysis tools. While that is helpful, it assumes that the interactions are simple and linear and that there are no cross dependencies. System-level models and systemic approaches are needed to address complex system behavior and cross-domain integration issues. Therefore, the simulation-based-acquisition architecture must incorporate systemic behavior. It is only then that the optimization at the system level becomes meaningful.

IMPLEMENTATION ISSUES

In addition to typical cultural and organizational issues that one must address in architecture strategies (Chadha 1997), a key implementation issue is the use of a top-down versus a bottom-up approach. We recommend a hybrid of the two based on the following observations:

- Bottom-up is relevant when there is little understanding of the problem.
- Top-down can completely address the problem (no items are missed).
- Bottom-up enables the problem to be addressed in small chunks.
- Top-down works in relatively stable environments; bottom-up is suitable for rapidly changing environments. The basic issue is whether one can completely solve the problem before the problem changes.
- Top-down is more efficient when there is a low probability of going astray and the emergent behavior due to component interaction is predictable and well understood. This implies that all the systemic interactions and causalities among components are known and accounted for (Perrow 1984).
- Bottom-up is more forgiving of mistakes and dead-end design paths. This is because mistakes are made on a smaller scale and are caught earlier and it is possible to change the system by rearranging components.
- Top-down is about "knowing it all" at the start; whereas, bottom-up is about "learning it" along the way.
- Top-down has slower design/implementation cycles; whereas, bottom-up has faster design/implementation cycles.
- Apply top-down at the conceptual level but apply bottom-up for details.

A hybrid of both approaches fits most real-world situations. Most projects start top-down before shifting to bottom-up at implementation, where the focus is on reuse of available, existing, or previously developed components. Strategies can also shift emphasis among top-down and bottom-up at multiple points in the development process. New or innovative developments are typically bottom-up (Kelly 1998). Strategies typically shift to top-down as processes mature, experience increases, and the problem is better understood.

SUMMARY

Virtual prototyping is critical to the cost-effective design of complex systems. New architectural concepts are emerging to support virtual prototyping of complex systems across company boundaries. When implementing virtual-prototyping architectures, realize there are limits to benefits achieved through commonality across distributed implementations and that soft/qualitative factors play an important role in architectural trade-offs. This paper outlined key architectural principles and concepts that improved the practicality of large-scale virtual-prototyping efforts. Many COTS tools and technologies are emerging to support implementation of these concepts. Effectively leveraging these tools and technologies will manage the scope of the effort while maintaining cost and schedule constraints. Bottom-up and top-down design approaches have advantages but a hybrid approach is best for most implementations.

REFERENCES

Alexander, C. 1964. "Notes on the Synthesis of Form," *Harvard University Press*.

Chadha, B. 1997. "A Federated PIM for Supply Chains," DH Brown Symposium.

Gamma, E., et al. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley.

Kelly, K. 1998. *New Rules for the New Economy*, Viking.

Perrow, C. 1984. *Normal Accidents*, Basic Books.

Senge, P.M., et al. 1994. *The Fifth Discipline Fieldbook*, Currency Doubleday.

Womack, J.P., Jones, D.T., Roos, D. 1990. "The Machine that Changed the World," *MIT Press*.

"The Common Object Request Broker: Architecture and Specification," Revision 2.0, OMG, July 1995, Updated July 1996.

BIOGRAPHIES

Dr. Chadha is the principal investigator on enterprise engineering, supply-chain integration, and process-improvement initiatives at Lockheed Martin Advanced Technology Laboratories. He leads the Smart Product Model architecture team for Lockheed Martin's DD 21 program. He is a member of the ASME Engineering Information Management Committee and Supply Chain Council, and he chairs Lockheed Martin's Product Data Management Interfaces Working Group. He received his Ph.D. in Mechanical Engineering from Georgia Institute of Technology.

Mr. Welsh manages the enterprise-technology organization, leading collaborative-engineering projects for ship systems, mission-level analyses, and logistics-supply-chain improvement at Lockheed Martin Advanced Technology Laboratories. He has over 20 years technical and managerial experience on enterprise software, systems engineering, and electrical engineering. He has a Bachelors degree in electrical engineering from Villanova University and a Masters degree in systems engineering from the University of Pennsylvania.