Spring 2011

Adding Executable Context to Executable Architectures: Enabling an Executable Context Simulation Framework (ECSF)

Johnny J. Garcia
Old Dominion University

ADDING EXECUTABLE CONTEXT TO EXECUTABLE ARCHITECTURES: ENABLING AN EXECUTABLE CONTEXT SIMULATION FRAMEWORK (ECSF)

by

Johnny J. Garcia
M.S. December, 2002, Florida Institute of Technology
M.B.A. December, 2003, Florida Institute of Technology
B.S. May, 1999, St. Leo College
B.A. May, 1999, St. Leo College

A dissertation submitted to the faculty of
Old Dominion University in partial fulfillment of the
Requirements for the degree of

DOCTOR OF PHILOSOPHY

MODELING AND SIMULATION

OLD DOMINION UNIVERSITY
May 2011

Approved by:

_____
Andreas Tolk (Director)

_____
Tom Pawlowski (Member)

_____
Charles Keating (Member)

_____
Frederic D. McKenzie (Member)

ABSTRACT

ADDING EXECUTABLE CONTEXT TO EXECUTABLE ARCHITECTURES: ENABLING AN EXECUTABLE CONTEXT SIMULATION FRAMEWORK (ECSF)

Johnny J. Garcia
Old Dominion University, 2011
Director: Dr. Andreas Tolk

A system that does not stand alone is represented by a complex entity of component combinations that interact with each other to execute a function. In today's interconnected world, systems integrate with other systems - called a system-of-systems infrastructure: a network of interrelated systems that can often exhibit both predictable and unpredictable behavior. The current state-of-the-art evaluation process of these system-of-systems and their community of practitioners in the academic community are limited to static methods focused on defining who is doing *what* and *where*. However, to answer the questions of *why* and *how* a system operates within complex systems-of-systems interrelationships, a system's architecture and context must be observed over time, its executable architecture, to discern effective predictable and unpredictable behavior.

The objective of this research is to determine a method for evaluating a system's executable architecture and assess the contribution and efficiency of the specified system before it is built. This research led to the development of concrete steps that synthesize the observance of the executable architecture, assessment recommendations provided by the North Atlantic Treaty Organization (NATO) Code of Best Practice for Command and Control (C2) Assessment, and the metrics for operational efficiency provided by the Military Missions and Means Framework. Based on the research herein, this synthesis is designed to evaluate and assess system-of-systems architectures in their operational context to provide quantitative results.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Andreas Tolk, for his continuous support, guidance and friendship throughout this work. His vision, enthusiasm and dedication to science, modeling and simulation, and education have been influential. There are no words to express my gratitude, admiration and affection for him. It has been a privilege to work under his supervision. I would like to thank my committee member, Dr. Tom Pawlowski, for his guidance and feedback at critical points in my research. I would like to thank my other committee members, Dr. Frederick Mckenzie and Dr. Charles Keating, for their valuable suggestions throughout this study. All together, my committee's insights enriched my dissertation immensely. I would like to thank all my family, friends and sponsors for their continuous encouragement. *"But the Lord stood at my side and gave me strength"* (2 Timothy 4:17). Without the grace of the Lord I could not have accomplished this work. I am grateful and indebted to my mother and father, Juan and Diana Garcia, for their love, throughout my life. Of course, I would be remiss if I did not mention the extensive support and love of my mother and father-in-law, Julie Phelps and Serafin Escobar. I would also like to commemorate my grandmother, Della Lopez, who was my rock throughout my life and told me to never give up; I truly miss her. Finally and most specially, this study is dedicated to my wonderful wife of 21 years, Lorena Garcia, who finds the best in me and stands by me with her endless patience, strength, friendship and love through the journey of life and to my wonderful twin

daughters (Hope and Faith) who have provided me continuous love, hugs and kisses when daddy was tired and worn from the process of this work.

TABLE OF CONTENTS

Page

LIST OF FIGURES

LIST OF TABLES

LIST OF EQUATIONS

1. INTRODUCTION

1.1. OVERVIEW

The U.S. Department of Defense (DoD) and other agencies and organizations deploy systems supporting mission critical operations. On the front end of system development, particularly during procurement, analysis and experimentation are often conducted to ascertain the effectiveness of a developing system to meet defined mission requirements. Supporting this task is the modeling and simulation (M&S) community which assists the overarching goal of the procurement community to evaluate a system's architecture before building the specified system. Current analysis techniques are performed using static evaluation of the system's architecture; in essence, these techniques merely evaluate the system in a controlled environment while examining the coherence and plausibility of the architecture's artifacts. These static evaluation processes answer who (entity) is doing what (function) where (component) (Banks et al. 1987; Balci, 1987).

Conducting an appropriate dynamic analysis of a system's effectiveness and performance in its intended operational environment often proves difficult since present approaches focus on technical and architecture systems (Maranzano et al., 2005) being represented in drawings, flowcharts, PowerPoint® presentations, and block diagrams. This tabletop analysis does not exhibit the characteristics of the executable architecture and thus limits the known and unknown system behaviors to only who is doing what and where. Systems supporting the critical missions of the DoD, whether developed for Battlespace Management, Intelligence-Surveillance-Reconnaissance (ISR), Force Protection, Service Management, Freedom of Movement, Medical Evacuation, or other operations within the DoD, are required to comply with the Department of Defense Architectural Framework (DoDAF), an architectural evaluation. Although DoDAF is considered state-of-the-art and represents the key cognizant analysis vehicle of the intended system, most of the requirements have been recognized and the possible situations are offered as given, potentially "outside the box" options (Levis et

al., 2000; Handley et al., 2000). DoDAF presently does not contain trade-off analysis, game theory projections, Monte Carlo simulation, or other complicated modeling analytical support tools (Charles & Turner, 2004). In its current state, DoDAF starts at a universal level (DoDAF V1.5 Vol. I and II, 2005) but fails to extrapolate the behaviors characteristic of the executable architecture. This lack of robust features and ability to accurately evaluate the architecture was noted by Levis (Levis & Wagenhals, 2006). Levis identifies this as a precise objective with no framework to accomplish this objective. He stated:

> *The derivation of an executable model of the architecture from the views and the associated integrated dictionary provides a basis for understanding the interrelationships among the various architecture products and establishes the foundation for implementing a process for assessing and comparing architectures (Levis & Wagenhals, 2000, p. 226).*

According to ISO/IEC 15939:2002, an attribute is a "characteristic of an entity that can be distinguished quantitatively or qualitatively by human or automated means" (2000, p. 154). Architecture attributes are important because they describe the properties of the system in a unique, distinguishing manner. Whether described granularly leaving little doubt which components are codified into the system's design, esoterically for confidentiality or a specific community's comprehension, or generally to ascertain primary requirements, architecture attributes establish the baseline of a system for mission or operational assurance.

Measurability of entities makes architectures ideally useful for monitoring and tracking many systems' engineering tasks. Bass, et al. (1998) used entities to measure systems architectures in making valuable decisions and tradeoffs in evaluating the architecture (p. 221-237). Although Bass's entities improved an organization's decisions affecting system development or acquisition, the context of external behaviors remain excluded from the evaluation process. Since systems are no longer islands to themselves, neglecting the effects of other systems could produce a variety of unintended or unwanted results.

How are systems modeled? The V-Model (Figure 1) is a systems development model designed to simplify the understanding of the complexity associated with developing systems (Forsberg, 2005; INCOSE, 2007).

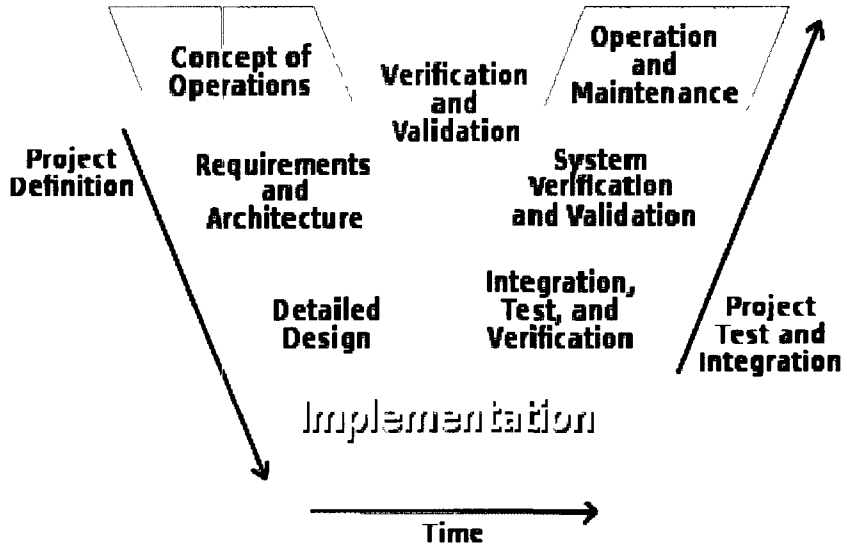


Figure 1: V-MODEL

In systems engineering, the V-model is used to define a uniform procedure for product or project development. The V-model is a graphical representation of the systems development lifecycle. It summarizes the main steps to be taken in conjunction with the corresponding deliverables within the computerized system evaluation framework. The "V" represents the sequence of steps in a project life cycle development. It describes the activities and results that must be produced during product development (Forsberg, 2005; INCOSE, 2007). The left side of the "V" represents the decomposition of requirements and creation of system specifications. The right side of the "V" represents integration of parts and their verification.

What is systems architecture? According to Zachman's Framework, an enterprise architecture framework provides a formal and highly structured way of viewing and

defining an enterprise, while systems architecture is described as "not systems architecture, but a set of them. Architecture is relative - what you think an architecture is depends on what you are doing" (Zachman, 1987). When considering how the product – the system represented by the system architecture – will be used, it is apparent that the system will exist in a dynamic environment in which it must address multiple, concurrent tasks. Today's state-of-the-art executable architectures do not effectively address how architectures are evaluated within the entirety of their context. In essence, the "why" and "how" architectures function in their intended environment or purpose before fielding remains unresolved.

Through conducted research, the concept of executable context was developed with the intent to model the systems architecture within a system's intended environment or its "context." The main objective or problem statement of this research is, "Can systems architecture be modeled within its operational and systems context? If so, does this lead to better decisions after the system is evaluated?" Figure 2 shows how current state-of–the-art systems architecture evaluation focuses on either the operational model or the systems model, rather than in a harmonized effort.

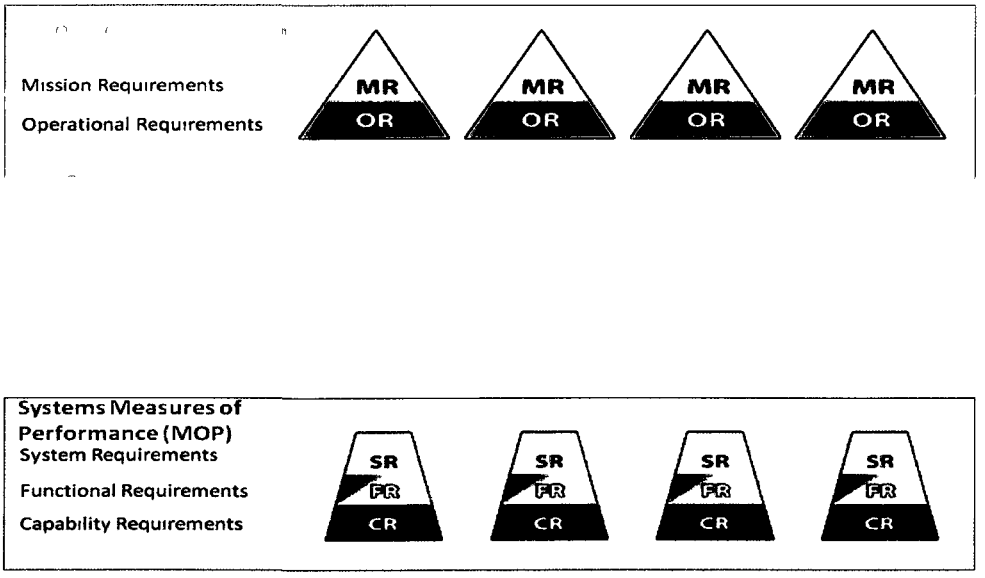


Figure 2: System and operational architecture disjointed evaluations

This research uses contributions from the disciplines of modeling and simulation (M&S) and systems engineering (SE) to functionally orchestrate the dynamic execution of operational and systems architecture to answer specific questions with the executable context simulation framework developed within this research effort.

1.2. RESEARCH METHOD

To aid in the understanding of theoretically-based research findings, it was necessary to test the state-of-the-art in systems architecture evaluation against theoretically-based challenges. Depending on the statistical results, this may include outcomes that combine effects of factors indirectly related to the systems architecture. Therefore, assessment of the systems architecture evaluation may prove to be difficult. These potential obstacles may be overcome by adding qualitative results to the quantitative outcome (Green, et al., 1987).

In the case of this thesis, the research method expands the research breadth and enlightens the more universal debate on system-of-systems architecture evaluation. In summary, this research strategy that comprises this thesis integrates quantitative and qualitative methods, or mixed methodology, with the intent to produce an intrinsic awareness of system behavior, capture a broader scope of how external elements affect systems behavior, and reduce potential risk imposed by elements undetected in the current static evaluation methods. In addition, the research strategy intends to develop a method for the M&S community to probe underlying issues imposed by external systems by using mixed-method analysis – defined as creative alternatives to traditional or monolithic ways to conceive and implement architecture evaluation.

1.3. RESEARCH OBJECTIVES

The intent of this thesis is to determine how to capture and execute the system in context. Defined in detail in section 2, evolving the static architecture evaluation process into systems context comprehension will use a systematic method to induce dynamic modeling. Each architectural capability will be identified and evaluated to

ascertain performance and system effectiveness, particularly concerning their operational context. The inclusion of the external environment influences how systems operate.

By including operational context within this protocol, the research will develop a method to support repeatable and measurable environments while producing a more reliable and representative systems architecture context. The circumstantial analysis of dynamic modeling support capabilities uses replaceable components that can be introduced or excluded to instantiate systems architecture capabilities and evaluate operational objectives. Key research observations are driven by these questions:

- What are the systems that are affected by this system?
- What are the systems that affect this system?
- What environment does this system operate in?
- Will this system execute within its intended environment as predicted?

1.4. RESEARCH APPROACH

The research approach is to develop a method to convert architecture products into an executable model and generates a federation of simulations that represents a system of systems. The research is based upon an examination of the systems' operational environments and operational mission threads. The findings of this research generalize this methodology and provide resources for the methodology to function with multiple frameworks and models. Further, the research explores how an executable context is defined based on theoretical and real-world operational examples. In summary, the research approach is directed to determining how or if the incorporation of the context leads to different decisions.

1.4.1. ACTIVITIES

The research activities of this dissertation were approached using four concrete steps. First, the theory was developed. Second, the theory was tested based on theoretical cases to address technical issues associated with the utilization of models

and simulations. Third, the research was built into a methodology to improve the management of related information. Fourth and finally, the methodology was used to develop a conceptual solution to provide quantitative results of architecture in an operational context.

1.4.2. IMPACT

By applying this method of architecture evaluation, the evolution of system-of-systems may be significantly impacted. It is hypothesized that dynamic, evaluated architectures will develop greater operational accuracy by providing more accurate and appropriate analysis of system-of-systems architectures. The developed method enables the application of system specific measures of performance based on system architecture specification to contribute directly to the operationally relevant measures of effectiveness required to evaluate the systems in their intended operational contexts. To achieve this, the research evaluated systems architectures for connectivity, performance, and information flow within their intended purpose of operation.

1.5. RESEARCH ORGANIZATION

This dissertation is organized in nine chapters. Chapter 1, *Introduction,* defines the research method, objectives, and approach to activities and their impact. Chapter 2, *Literature Research*, provides the literature review of related research, thus establishing the applicability of the research contained herein. Chapter 3, *Research Challenge and Problem Set*, identifies the gap this research intends to close through the advancement of the current state of system-of-systems architecture evaluation and research accuracy. Chapter 4, *Research Leveraged Methods,* provides an overview of how the research method advances the state-of-the-art within existing methods regarding their applicability to system-of-systems architecture evaluations. Chapter 5, *Method Development and Overview,* details the proposed theoretical solution and method to establish the academic research foundation, which includes research generalization of other architecture frameworks. Chapter 6, *Bounding the Research: Executable Context Engineering Element Examples*, bounds the research, experimental results, and

synthesis of the executable context engineering elements. Chapter 7, *Conclusion*, provides conclusions for the theoretical, methodology and solution of the research. Chapter 8, *Extensibility of the Research*, identifies future research areas for extending and enhancing the executable context method for portfolio management and other domains for system-of-systems, conclusions and open research directions. Finally, chapter 9, *References*, provides all references described within this dissertation.

Table 1 below describes the organization in better details in relation to theory, method and solution. This table aligns what elements are used to aid in the theory of the research, the development of the method and how the theory and the method were used to develop a solution to the problem. These elements will be used to define each section of the document.

	State of the "Art"	**Data**	**Research Findings**
Theory	Literature: 1. Systems Engineering 2. System-of-systems 3. Architecture Frameworks 4. Architecture Evaluation	Research	Gap Identified
Method	1. DEVS Unified Process (DUNIP) 2. Method Architecture Validation (MAVS) 3. NATO Code of best practice (NCOBP) 4. Mission to Means Framework (MMF)	Static Information	Executable Architecture defined
Solution	1. Discrete Event System Specification (DEVS) 2. JAVA DEVS 3. Department of Defense Architectural Framework (DoDAF) 4. Executable Context Simulation Framework (ECSF)	Federated models – Modeling & Simulation	Quantitative Knowledge "Executable Context" defined

Table 1: Research Alignment

2. LITERATURE RESEARCH

A system's context, as defined by Buede (2000), is a set of entities that interact with other systems via the system's external interfaces. In Figure 3, Buede (2000) depicts where the external systems can impact the system and whether or not the system impacts the external systems. A system in Buede's (2000) depiction below may function by providing some context to an external source, consume other system's resources, or interact with the external system bi-directionally. Buede (2000, p. 38) further defines that "the entities in the system's context are responsible for some of the system's requirements as it applies to the external systems." Therefore, Buede's (2000) context definition incorporates that set of entities which support the interaction of a system with all other external systems.

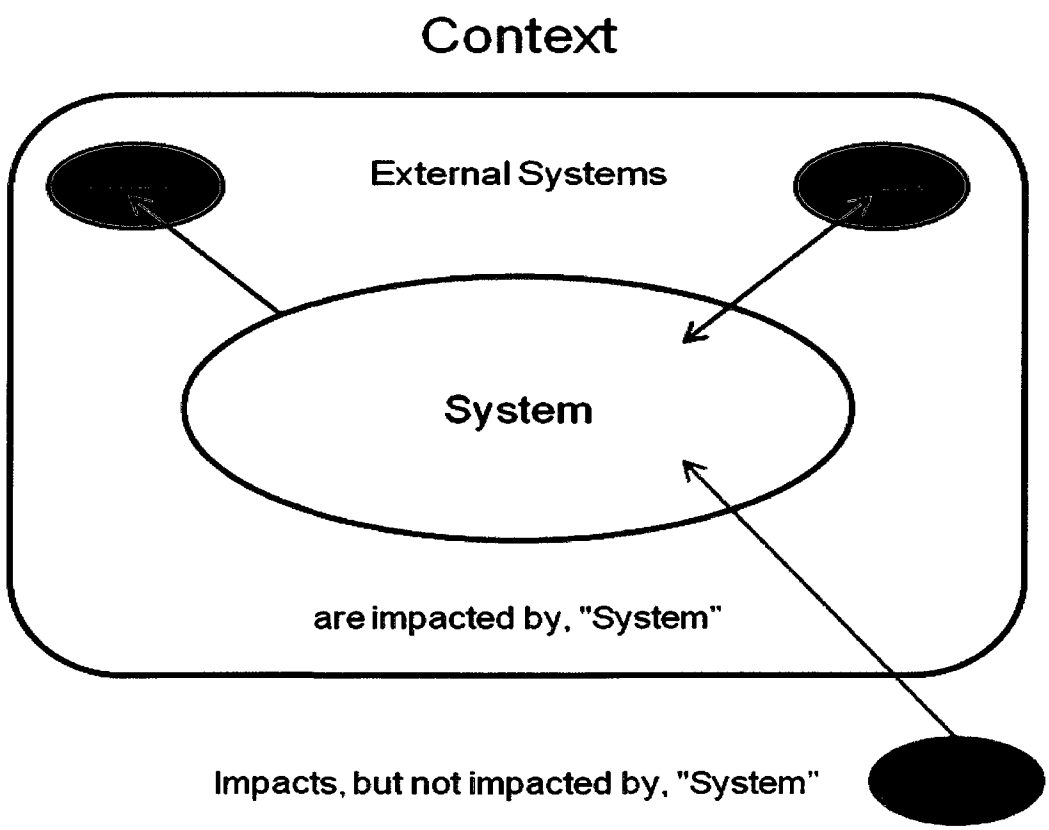


Figure 3: Buede's depiction of a system's "context" (Buede, 2000, p. 38)

A system's environment context attribute is dependent on the environment in which the system and its components exist (Crnkovic & Larsson, 2004). All systems reside within a context, and the context, to include those components of the environment, needs to be defined to aid in the evaluation of the system's operation. Levis, on the other hand, describes the context of a system as a set of entities that can impact the system but cannot be impacted by the system (Levis, 1993, p. 2-6). Levis's (1993) statement is true for a stand-alone system but not for today's interconnected environments that are amalgamations of many systems interacting in the modes described by Buede (2000) and are a part of a system-of-systems paradigm.

Leveraging the ideas of a context to define a system-of-systems model requires two methodologies for validating architecture: an information paradigm of evaluation and a knowledge paradigm for architecture evaluation. In Zachman's framework, systems architecture is described as "not systems architecture, but a set of them". Zachman developed six interrogatives - who, what, where, when, how, and why (1987) to define architecture element representation. Sage and Rouse (1999) expanded these interrogatives into two groups. One group relates to information (who, what, where and when. The second group relates to knowledge (why and how). This framework distinguishes between those elements that relate to information - *who* (people), *what* (entities), *where* (locations), *when* (time) - and those that relate to knowledge: *how* (functions) and *why* (purpose).

According to Russell Ackoff (1989), a systems theorist and professor of organizational change, the content of the human mind can be classified into five categories:

- Data
- Information
- Knowledge
- Understanding, and
- Wisdom

Ackoff (1989) states, "Data is raw. It simply exists and has no significance beyond its existence (in itself)." It can exist in any form, usable or not, and does not have

meaning in and of itself. Information is data that has been given meaning by way of relational connection. This "meaning" can be useful but does not have to be. Knowledge is the appropriate collection of information, such that its intent is to be useful. Knowledge is a deterministic process. When someone memorizes information, they have amassed knowledge. This knowledge has a useful meaning to that person, but it does not provide for, in itself, integration such that it would infer further knowledge. If integration of meaningful information and knowledge would infer further knowledge, systems that have an understanding of context may behave more reliably because they can synthesize new knowledge, or minimally, new information from what is previously known and understood. Understanding context can build upon currently held information, knowledge, and comprehension itself. Systems, in essence, exhibit understanding in the sense that they are able to synthesize new knowledge from its context. From these syntheses of information and knowledge, systems' architectural evaluations become information-based paradigms.

The emphasis of this research focuses the information-based evaluation paradigm, based on a body of knowledge, to the development of an executable context for systems architecture evaluation. This information-based approach for evaluation based on knowledge is desirable for systems architecture evaluation. In today's engineering environment, architectural evaluation is needed to support collaboration among designers, programmers, program managers, and stakeholders who will procure, test and ultimately use such systems.

Buede (2000) describes information as data in context. Knowledge is applicable information in procedural form (Polanyi, 1998). However using today's architecture evaluation methods, knowledge-based evaluation is not yet possible. The questions *how* and *why* a system acts must become part of the evaluation, otherwise referred to as knowledge-based evaluation as depicted in Figure 4. Knowledge-based evaluations include mission requirements (MR), operational requirements (OR) and external systems (ES) within the system's architectural definition. Figure 4 illustrates how executable context (EC) enables conditions under which architectures can be

experimented with and evaluated. The information interrogatives (*who*, *what* and *where*) are composed in the context of the system (the MR, OR and ES). An executable architecture defines the *when.* All metrics applied on these levels measure the performance of system components or sub-systems. The method developed within this research enables a system to be modeled in the environment that enhances the ability to answer *why* and *how* the architecture will be executed in its intended environment or for its intended purpose (the system-of-systems). In this environment, the effectiveness of the system in the operational context is measured by measures of effectiveness (MOE) and measures of performance (MOP).

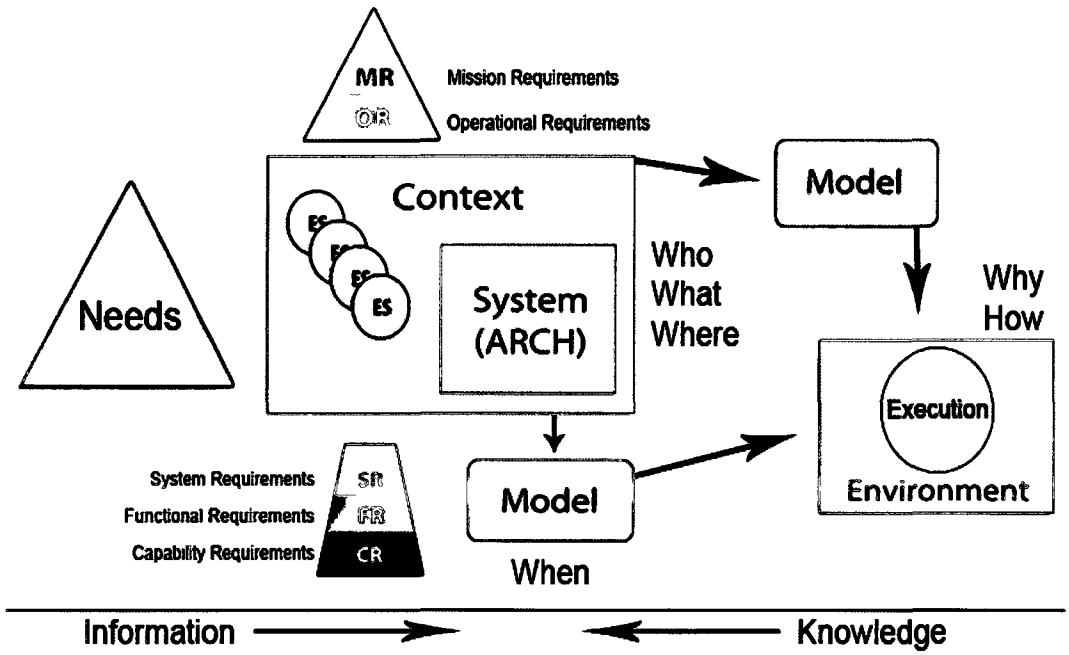


Figure 4: Executable context (EC) as it relates to knowledge-based evaluation

As emphasized in this research, such architectural evaluation and the resulting products must be completely dynamic to support these collaborative dialogs and to allow stakeholders to accurately understand the intended system function and its

intended purpose within the context of its environment. Since system governance is typically tied to limited resources, early detection and awareness of risk that could affect a stakeholder's operations should benefit from executable context architecture evaluation. This research introduces a new method to employ knowledge-based evaluation of systems architectures. While current approaches evaluate measures of performance on the tactical/system's level and measures of effectiveness on the operational level independently, the framework developed here allows immediate use of the system performance based on the system's specification and the use of it in the operational context to contribute to the measures of effectiveness. As such, all six identified interrogatives – who, what, where, when, why, and how – can now be evaluated in one common framework.

The next section provides state-of-the-art in other disciplines that bound the research method: systems engineering, architecture evaluation, system-of-systems, system architectures, architectural frameworks and executable architectures.

2.1. STATE-OF-THE-ART IN SYSTEMS ENGINEERING

Systems engineering focuses on the engineering of large-scale, complex systems (Sage, 1992). First and foremost, systems engineering is a trans-disciplinary management technology (Sage, 2002). The term systems engineering can be traced back to Bell Telephone Laboratories in the 1940s (Schlager, 1956) and, according to Hall (1962), is a way to identify and manipulate the properties of a system as a whole, which in complex engineering projects may greatly differ from the sum of the parts' properties. Hall's perspective motivated the Department of Defense, NASA, and other industries to apply the discipline of systems engineering (Hall, 1962).

As systems and their complex relationships grew, it was no longer possible to rely on design evolution to improve upon a system since the existing tools were not sufficient to meet growing demands on architecture evaluation (Sage, 1992). An evolution of systems engineering emerged comprising the development and identification of new methods and modeling techniques. Modeling aids in better comprehension of engineering systems as they grow more complex. When it was no

longer possible to rely on design evolution to improve upon a system and the existing tools were not sufficient to meet growing demands, new methods began to be developed that addressed the complexity directly. The evolution of systems engineering, which continues to this day, comprises the development and identification of new methods and modeling techniques. These methods aid in better comprehension of engineering systems as they grow more complex. Popular tools that are often used in the systems engineering context were developed during these times.

2.2. STATE-OF-THE-ART IN ARCHITECTURE EVALUATION

During architecture evaluation, stakeholders strive to verify the requirements of the system. In addition to enhancing confidence that the architecture will meet the demands placed on it, the inclusion of the right evaluation components during this phase can help generate confidence that the architecture will be able to support its intended purpose. Architecture assessment involves thought experiments – modeling and walking through scenarios that exemplify requirements – as well as an expert assessment that identifies gaps and weaknesses in the architecture as described in "Architecture Reviews: Practice and Experience" and "Best Current Practices: Software Architecture Validation" (*Best Current Practices*, 1990; Marazano, et al., 2005, pp. 34-43). Just as a system architect can not overlook such contextual factors as the network, security requirements, hardware and systems standards, the architect cannot overlook the context as defined in the research of the system. The key technical considerations alluded to by "system fit to context" have to do with interoperability, consistency, and interface with external systems. However, there are considerations to be factored into architectural evaluation and choices to fit within the development organization's culture and capabilities.

Architecture evaluations (AE) minimize duplicity and, with the help of high performance scalable designs, facilitate easy formation of new evaluations. AEs can produce a number of definite evaluations and enable a new understanding of evaluation failures in relation to the capability requirements. AEs are valuable in the identification of the types of applicable, accurate evaluation of data sources. AVs produce a

standardized flow identifying a set of required parameters for the sub-process evaluation as well as the accessibility of result data to perform data evaluation. AEs offer a framework for the performance of general processing needed for the evaluation majority flows by maintaining the evaluation subroutines' flexibility.

2.2.1. EVALUATION OF THE ARCHITECTURE

According to Bredemeyer, architects make their best effort to fulfill the requirements on the system throughout the evaluation phase of the architecture using an external architecture team to provide an objective evaluation of the architecture (1999). Evaluation of the architecture includes "thought experiments," modeling and walking through scenarios to illustrate the requirements as well as evaluation by specialists to identify architectural gaps and limitations based on their experience (Rechtin, 1991; Seliger, 1997).

Another vital part of architecture evaluation is the improvement of prototypes or proofs-of-concept. This is a more realistic, effective method of determining the future success of the architecture as it tests the basic version of the architecture when it is ready to implement. The architecture evaluation process is accomplished iteratively, with multiple cycles through requirements, structuring, and evaluation. This method yields the most control upon architecture specification but is normally complicated with the issues of organization (e.g., the "Not Invented Here "(NIH) syndrome) that decrease or even completely restrain the use of the architecture (Bredemeyer, 2007).

According to Bredemeyer's research, the process of evaluating architecture specification is the most difficult to accomplish (Bean Architect, 2007). To enable a valid outline of the architecture (who, what and where), Bredemeyer broke the process into sub-phases, along the outline of the architecture, to aid in the evaluation.

Meta-Architecture: To aid in making decisions, the visualization of the architecture is originated first. It is good to explicitly assign research time to generate ideas in documented architectural styles, dominant designs, patterns, reference architectures, or other architectures within the context of the system.

Conceptual architecture: The architectural system is then reduced to the components and the responsibilities of each component while considering the interrelation of various components. The objective of the conceptual architecture is to concentrate on suitable system decomposition without focusing on the requirement specification and information type. In addition, conceptual architecture is a helpful medium to communicate regarding architecture to the non-technical stakeholders, i.e. marketing and management departments (Bredemeyer, 2010).

Logical architecture: The conceptual architecture creates the preliminary point for the logical architecture. Logical architecture is possibly developed and also distinguished in the architecture establishment period. Developing the system activities as dynamic capabilities is a helpful method in the architect's thinking process regarding the component's interfaces and responsibilities. Component specifications influence the architecture.

Another important part of architecture evaluation is the development of prototypes or proofs-of-concept. Taking a skeletal version of the architecture all the way through to implementation, for example, is a highly effective method of evaluating aspects of the architecture (Bredemeyer, 2010).

This research used the latest version of the Department of Defense (DoD) Modeling and Simulation Glossary, which defines evaluation as "the process of determining the degree to which a model (architecture) or simulation is a faithful representation of the real world from the perspective of the intended uses of that architecture" (Defense Modeling and Simulation Office, 1997 p. 162). Evaluation, as described by Banks, et al., demonstrates that a computerized model satisfies the simulation objectives and requirements with sufficient accuracy within its domain of applicability (1987).

In these definitions, the terms "real world" and "domains" refer to the entities needed to enable an executable context to answer the interrogatives *how* and *why*. Prior research conducted by Levis, Mittal, and others enabled executable architectures

to answer the *when* of the six interrogatives, and architectural frameworks provide the means to answer the *who*, *what,* and *where* (Mittal, 2006; Levis, & Wagenhals, 2006).

The current state of the art in architecture evaluation has shown that evaluation of systems architecture is based on model developer interpretation to evaluate the operational architecture and systems architecture artifacts independent of each other. This research enables the integration of the independent models into a common method that allows harmonization of system and operational architecture as an executable. This research also allows the resulting artifacts be federated into an executable context that represents all external systems and can be initialized with the elements describing an operational scenario, allowing relevant measures of performance on the system level and measures of effectiveness on the scenario level to be derived from operational requirements while using standard simulation architectures environments and common frameworks.

2.3. WHAT IS A SYSTEM-OF-SYSTEMS?

There are many definitions of system-of-systems (SoS) depending on the application area and focus (Maier, 2005, p. 3149-3154; Carlock, et al., 2001, p. 242-261; Sage, et al., 2001, p. 324-345; Gideon, et al., 2005; Keating, Rogers, Unal, Dryer, et al. p.36; Keating, 2005). Popper, Bankes, Callaway and DeLaurentis (2004) describe SoS as a collection of task-oriented or dedicated systems that pool their resources and capabilities together to obtain a new, more complex, 'meta-system' which offers more functionality and performance than simply the sum of the constituent systems.

Several combinations of characteristics are observed in SoS (Bar-Yam, et al., 2004):

- Operational independence of elements
- Managerial independence of elements
- Evolutionary development
- Emergent behavior
- Geographical distribution
- Heterogeneity of systems
- System of networks

SoS studies are interdisciplinary and span through the study of architecting as well as various modeling and simulation techniques including network theory, systems theory, uncertainty modeling, agent-based modeling, and object-oriented simulation.

This research emphasizes the use of SoS to define measures, operational systems architectures and visual tools for capturing systems and operational requirements, and decision and operational analysis of external systems which are needed to aid in the development of the SoS to establish the context of the system being evaluated

2.4. STATE-OF-THE-ART IN SYSTEMS ARCHITECTURE

Systems architecture is necessary to describe the structure of a system. Every system has an architecture, whether it is explicitly or implicitly designed and documented. Architecture has many definitions. The International Council on Systems Engineering (INCOSE) defines systems architecture as, "the arrangement of elements and subsystems and the allocation of functions to them to meet system requirements" (INCOSE, 2006 p 9). IEEE 1471 defines architecture as the "fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution" (Institute of Electrical and Electronics Engineers, 2000). Merriam-Webster defines systems architecture as "a conceptual design that characterizes the structure and/or behavior of a system" (Merriam-Webster, 2003). Buede defines systems architecture as a way to "provide the foundation for developing and evaluating engineered system of systems" (Buede, 2000, p. 38).

Systems architecture includes the process for generating a functional, physical and operational architecture from a top-level operations concept. A state-of-the-art robust architecture exhibits an optimal degree of fault-tolerance, backward compatibility, forward compatibility, extensibility, reliability, maintainability, availability, serviceability, usability, and such other attributes as necessary and/or desirable. Systems architecture is a process for planning and building structures and systems to respond to a given need (Rechtin & Maier, 1997). The set of relations, which the

architecture describes, can be expressed in various ways such as software, hardware, organizational management, or knowledge representation.

The essence of system architecting is structuring by bringing form to function, by bringing order out of chaos, and by converting partially formed ideas of a client into a workable conceptual model. In systems architecting, the alternative architectures are large and the selection is not easy. Therefore, the systems architecting process focuses on balancing the customer needs, fitting the interfaces of system components, and compromising among the key system attributes, such as cost, risk, schedule, and performance. Systems architecture is concerned with the internal interfaces among the system's components or sub-systems and the relationship between the system and its external environment. It is a representation because it provides the elements comprising a system, the relationships among the system elements, and the rules governing the relationships. It is also a process because a sequence of steps is necessary to design or change the architecture of a system.

Systems architecture can best be described as a representation of an existent or "to be created" system and the process and discipline for effectively implementing the design(s) for such a system. The set of relations (that is, embedded information) that architecture describes may be expressed in hardware, software, or other application. Although the words between these definitions are somewhat different, the concept behind architecture is consistently described as organizing a system into constituent parts as specified through requirements to satisfy a desired goal. One challenge when discussing architecture is to understand what part of the architecture is under discussion and establishes the need for an executable context of the systems architecture. Architecture frameworks help in the organizing of architectural information.

2.5. STATE-OF-THE-ART IN ARCHITECTURAL FRAMEWORKS

Architecture frameworks improve understanding by providing systematic approaches to architecture development. However, many aspects of architecture remain unambiguous (Tang, Han, & Chen, 2004). IEEE 1471:2000 defines the primary

goal of architectural frameworks as an indication of "what information regarding architecture is important to be captured in architecture descriptions and to provide means for capturing this information" (Institute of Electrical and Electronics Engineers, 2000). Architectural frameworks guide the selection of what information is relevant for this purpose and trigger the architecture description.

An architecture framework provides a consistent approach for standardizing, planning, analyzing and modeling these entities for this research. Several architecture frameworks have been published for this purpose. Activities defined in these architecture frameworks vary, as do their outcomes. After examining different architecture frameworks and methods for architecture evaluation such as *A Framework for Classifying and Comparing Software Architecture Evaluation Methods* (Barbar, M.A., et. al., 2004, pp. 309-318), the *IEEE Recommended Practice for Architectural Description of Software-Intensive System* (Institute of Electrical and Electronics Engineers, 2000), and *A Comparative Analysis of Architecture Frameworks* (Tang, Han., & Chen, 2004), this research leverages the Department of Defense Architecture Framework (DoDAF) as its fit-for-purpose: an adaption of specific principles to be applied to all programs for standardized language and presentation of the architecture framework to ensure architecture solutions are appropriate for the DoD.

2.5.1. DEPARTMENT OF DEFENSE ARCHITECTURE FRAMEWORK (DoDAF) PROVIDES THE *WHO, WHAT* AND *WHERE*

The Department of Defense mandates that DoDAF be adopted to express high-level system and operational requirements and architectures (DoDAF Working Group, 2003). DoDAF is the basis for the integrated architectures mandated in DOD Instruction 5000.2 (2003) and provides broad levels of specification related to operational, system and technical views (Chairman, Joint Chief of Staff (JCS) Instruction 3170.01D, 2004; Chairman, JCS Instruction 6212.01D, 2006). DoDAF and other DoD mandates pose significant challenges to the DoD system/operational architecture development and testing communities because DoDAF specifications must be evaluated for compliance with requirements and objectives, even though they are not expressed in a form

amenable to such evaluation. However, a DoDAF-compliant system does not have the necessary information to construct high-fidelity simulations (Mittal, 2006; Levis, & Wagenhals, 2006). Such simulations become, in effect, the executable architectures referred to in the DoDAF document or in the context of this research, the *when*.

DoDAF is mandated for large procurement projects in the Command and Control domain, but its use in relation to M&S is not explicitly mentioned in the documentation (Atkinson, K., 2004, p.8; Atkinson, 2010; DoD Metadata Registry and Clearinghouse, 2004). Thus, an opportunity has emerged to support the translation of DoDAF-compliant architectures into models that are of sufficient fidelity to support architectural evaluation in simulation environments. Operational views capture the requirements of the architecture being evaluated and system views provide its technical attributes. Section 6.2 will provide greater detail on how DoDAF was used within an executable context as related to this research. Together, these views form the basis for semi-automated construction of the needed models for an executable context.

2.6. STATE-OF-THE-ART IN EXECUTABLE ARCHITECTURES

Although executable architectures are rooted in several years of research on transforming modeling languages into executable artifacts, the focus of this research lies on those approaches that emphasize the operational aspect of the use of the defined systems, in particular in the military context (although not limited to this context in its implications). To these predecessors of executable architectures belong, in particular, the approaches on Architecture Description Languages (ADL) (Clements, 1996). The work described by McKenzie, Petty, and Xu (2004) shows an application thereof to improve federation design. Other related work deals with executable Universal Model Language (eUML) (Mellor, 2002). All these approaches are useful but do not focus on the evaluation of tactical performance and operational effectiveness and efficiency.

According to Levis, executable architecture is described by the DoDAF as "utility of dynamic and energetic simulation software to estimate architecture models" (Levis, & Wagenhals, 2006). Levis emphasizes the assessment of the executable model completely to define and understand the dynamic features of the system's needs and

executable model. To maintain this practice, the executable model has supported the use of Colored Petri Nets (CPN) (Levis, & Wagenhals, 2006).

Andrew Zinn declared that, as per Levis, Holly and Handley, the highlighted Petri Nets must manufacture an executable model that aligns all the sequences in multiple views or static models into a single model (2004). Lee, et al. used Systems Engineering standard EIA 632 as the process for engineering a system and applied the DoDAF six-step guidelines to develop architecture templates to assist in the project (Lee, et al., 2005). Executable contexts (EC) – the method developed herein – also use the DoDAF six-step guidelines with some enhancements for architecture evaluation (Garcia, 2010). Following Pawlowski's proposal of the Executable Architecture Methodology for Analysis (EAMA), others have discussed, designed, and proposed different approaches to deal with executable architecture issues (Pawlowski, et al., 2004). For example, Executable Architecture Analysis Modeling Method (EAAM) will enable an organization to conduct dynamic, persistent, extensible, measurable, repeatable, and interactive testing (Garcia, 2009). This research used and leveraged a number of research activities that support *who* (people), *what* (entities), *where* (locations), *when* (time), and refer to information within the architecture as described by Levis, Mittal, and others (Levis, & Wagenhals, 2006; Mittal, 2008). This research enables knowledge assessment of *how* (functions) and *why* (purpose) when dealing with evaluation of the architectures.

As research objectives and goals were established, research findings revealed a necessity to consider a range of system interoperability factors and environments while making crucial decisions in executable architecture development. To respond to these factors, the research process used recent interoperability technologies and currently improved adaptations effectively to incorporate executable architecture with the objective environment. Methods of system interoperability target web services and other applicable standards of World Wide Consortium (W3C) (www.w3c.org) as much as possible to ensure that the communications are compatible with the remote system and that evaluation is accessible (Austin, et al., 2004).

3. RESEARCH CHALLENGE AND PROBLEM SET

This research focuses on contributions to the relevant body of knowledge that will enable the use of knowledge to evaluate architectures as defined in Figure 5 below. This research will incorporate the *how* and *why* or *knowledge-based evaluation* through executable context. This research intends to show how knowledge based evaluation enhances system-of-systems into an executable context that represents all external systems and can be initialized with the elements describing an operational scenario and allows relevant measures of performance on the system level and measures of effectiveness on the scenario level to be derived from operational requirements.

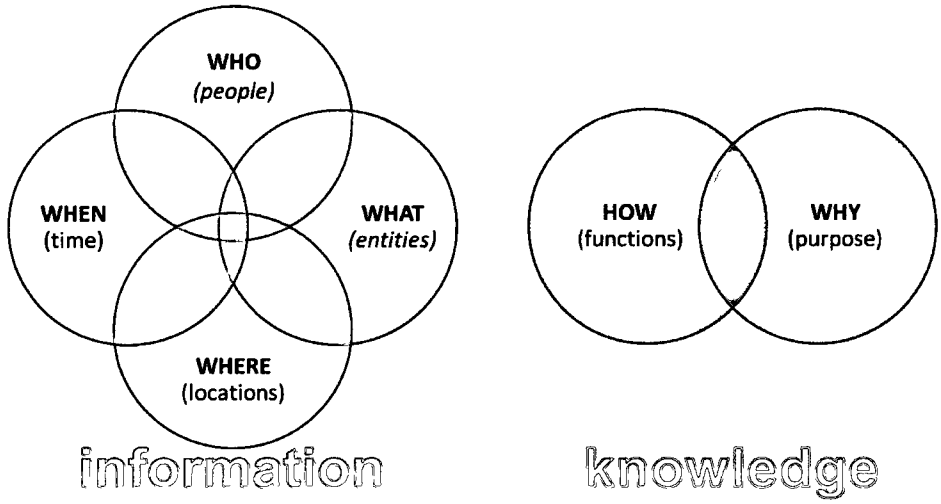


Figure 5: Research intentions from information to knowledge

3.1. WHY IS THIS A GAP?

Executable context (EC) provides the ability to conduct knowledge-based evaluation. Current system-of-systems architecture evaluations are limited to information-based schemas identifying routine requirements such as connectivity among nodes in the architecture. Information-based evaluation identifies *who, what,* and *where*. An executable architecture is defined as a dynamic model of the sequencing

of activities performed at operational nodes by roles (within organizations) using resources or systems to produce and consume information (Pawlowski et al., 2004). This research will provide a means to assess evaluation of systems architecture performances to meet its intended purpose. However, as stated in the literature review, executable architectures are only proven to depict the *when*.

Knowledge-based analysis is critical to assess the system-of-systems against the operational conditions expected by the mission requirement. To determine this effectiveness, it is necessary to employ an architectural representation that one can execute in a simulation environment so that system performance and, subsequently, its effectiveness and evaluation can be measured within its intended environment or executable context.

The challenge – otherwise identified as the research method developed by this research to aide in closing the gap – is how this research can be used to evaluate a system's performance and effectiveness when operating in its operational environment. Executable context closes this gap by incorporating the *why* and *how* or *knowledge-based evaluation*.

The correlated challenge is the use of measures of performance (MOP) – measuring system performance regarding the interrogatives who, what, where, and when – based on executable architecture systems specifications in direct support of measures of operational effectiveness (MOE) – measuring the operational system contributions in the context of operations regarding the interrogatives why and how.

3.2. SPECIFIC RESEARCH OBJECTIVES

This research is intended to develop a method and implement a supporting framework based on executable architectures, the *NATO Code of Best Practice (NCOBP) for C2 Assessments* and the Missions and Means Framework to enable evaluation of system-of-systems architectures using an executable context.

In particular, the following questions have to be addressed:

- What is an appropriate approach to make a system specification that is available in the form of a system architecture executable?

- How can the resulting artifact be federated into an executable context that represents all external systems and can be initialized with the elements describing an operational scenario?
- How can the relevant measures of performance on the system level and measures of effectiveness on the scenario level be derived from operational requirements?

3.3. PROPOSED SOLUTION THEORY AND METHOD

The intent of the research explored in this dissertation is to develop a method and framework that supports the evaluation of a system-of-systems architecture within its operational context. This process creates a systematic method to evolve the current information-based architecture evaluation process into a knowledge-based executable context method. This new method will identify architectural capabilities and provide measures of the performance and effectiveness of the system-of-systems.

The proposed method starts with the system architecture and utilizes appropriate methods, as identified in the first step of the research, to generate an executable architecture that supports access to all specified details. Next, this executable artifact is modified into a federate. Using methods defined by the operational community, operations relevant to the user of the new system scenarios and metrics for measures of performance and operational effectiveness are identified. Using standardized engineering methods, a federation to execute the system's architecture in the operational context delivering the required results for the identified metrics is developed and executed.

Figure 6 shows how entities, interactions and the conceptual model enable evaluation elements needed to develop the blueprint of the targeted valid systems architecture as described above. In the figure below, these elements are mission requirements (MR), operational requirements (OR), system requirements (SR), functional requirements (FR), capability requirements (CR), and external systems (ES). Following the method as specified in Buede (2000), all five requirements groups are

derived in collaboration of customers, users, and engineers from the operational goals envisioned to be supported or enabled by the system to be developed.

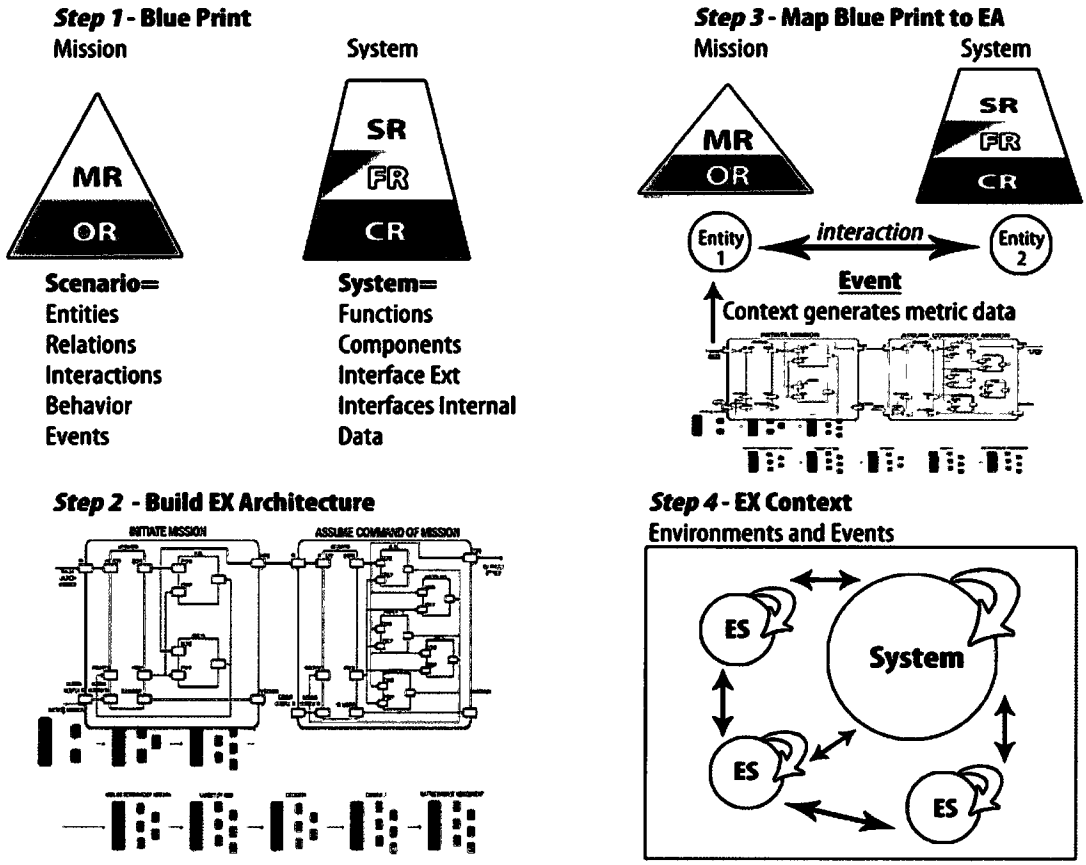


Figure 6: EC's four steps for evaluating targeted systems

The figure shows EC's four steps for evaluating targeted systems as:

- Develop context blueprint - Identify metrics
- Build Executable Architectures (EA)
- Develop Executable Context (EC) - Federate the EA and EC
- Execute context and observe quantitative metrics

3.4. EVALUATION METHOD

Evaluation is a "Proof of Specified Performance" (MDA, 2008). It follows that a validation process requires specification, performance, and means of proof. Furthermore, *DoD Modeling and Simulation Glossary* defines evaluations as "the process of determining the degree to which a model (architecture) or simulation is a faithful representation of the real world from the perspective of the intended uses of that architecture" (p. 162).

To ensure the EC method meets the criteria for architecture evaluation within the M&S community, the following valid practices, specifications, frameworks and methods are employed:

- Extend the use of DoDAF modeling to include provisions for M&S through the Discrete Event System Specification (DEVS) Unified Process (DUNIP) to enable DoDAF to become the executable architecture
- Apply Distributed Simulation Engineering and Execution Process (DSEEP) IEEE Std. 1730-2010
- Leverage the NATO Code of Best Practice (NCOBP)
- Map DoDAF to the Mission to Means Framework (MMF)

The next section will give more details for these methods and how they were applied in the context of this research to enable the envisioned framework. The following table summarizes the appropriate methods to answer the questions identified earlier.

Research Question	Identified Method
What is an appropriate approach to make a system specification that is available in form of a system architecture executable?	• Application of DUNIP • JAVA DEVS
How can the resulting artifact be federated into an executable context that represents all external systems and can be initialized with the elements describing an operational scenario?	• Extending DUNIP to result in a federate • Apply DSEEP to identify federates that can represent external systems and operational context • Develop federation
How can the relevant measures of performance on the system level and measures of effectiveness on the scenario level be derived from operational requirements?	• Apply MMF to identify relevant scenarios • For each scenario, apply MMF to identify MOP and MOE • For each MOP and MOE define data access points and data collection • For each relevant scenario, conduct simulation experiments

Table 2: Research Question and Identified Methods

These methods ensure and the resulting framework enables that all measures of performance on the system level are computed within the user relevant operational context based on the system's specification and contribute directly to the operational efficiency. Furthermore, the resulting level of detail allows for specific evaluation of system interactions in the context of being part of the operationally specified system-of-systems so that detailed analysis of system behavior in the operational context becomes observable.

4. EVALUATION OF EXISTING METHODS REGARDING THEIR APPLICABILITY

Best practices represent the current conventional wisdom applied to a particular condition or circumstance with the expectation of the result being the more effective of any other previous method, technique, activity, process, etc. Leveraging repeatable methodologies creates operational and system model harmonization for resulting artifacts to be federated into an executable context, as identified in the literature research, than what has historically been done.

4.1. NATO CODE OF BEST PRACTICES

When architectures are used to define new systems, the evaluation phase needs to show effectiveness and efficiency in the DoDAF context, often referred to as "fit-for-purpose," meaning the best solution for a given problem (DoDAF Working Group, 2003; DoD Instruction, 2003). In order to support these methods, metrics are needed.

The *NATO Code of Best Practice (NCOBP) for Command and Control (C2) Assessment* states that a "proper set of scenarios [use cases] is critical to assessment." It notes that scenarios should "consist of four elements – a context, the participants, the environment and the evolution of events in time." It notes that "the purpose of scenarios is to ensure that the assessment is informed by decision maker planning assumptions and the appropriate range of opportunities to observe the relevant variables and their interrelationships" (2002).

The NCOBP is designed to facilitate the transition from C2 theory (i.e., the C2 Conceptual Reference Model) to operational practice. The *NCOBP for C2 Assessment* established an operations research method that recommends best practices for the structure of architecture evaluation. Since 2007, the NCOBP has been adopted as a standard within the Joint Staff and Office of the Secretary of Defense (OSD) Networks and Information Integration (NII).

Figure 7 shows the structure of the NCOBP processes and their main domains to aid in problem formulation and analysis.

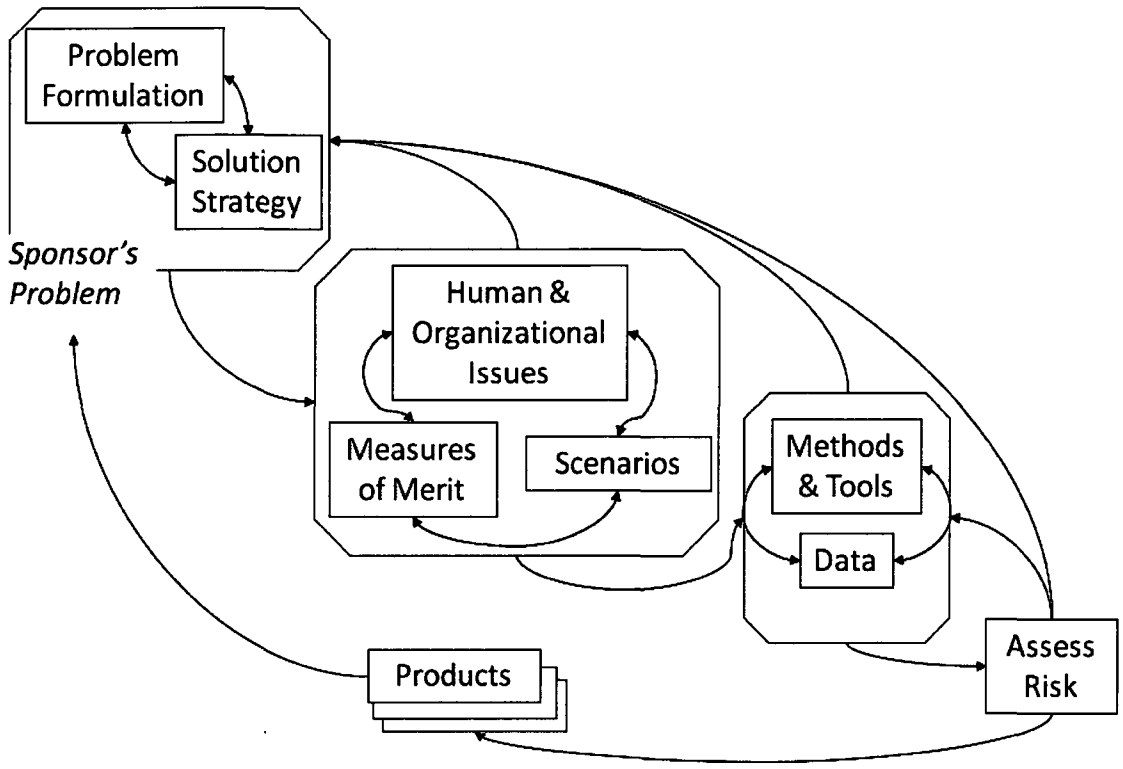


Figure 7: NCOBP problem formulation

To conduct this research, the NCOBP problem formulation was modified for EC to aid in architecture evaluation. EC problem formulation describes how the process of system-of-systems architecture evaluation goes from the context of a system (as described earlier in Buede) to the EC defined in the previous section. Figure 8 shows how the interactions of all entities enable evaluation in an EC that supports evaluation of effectiveness and efficiency guided by an accepted assessment solution.

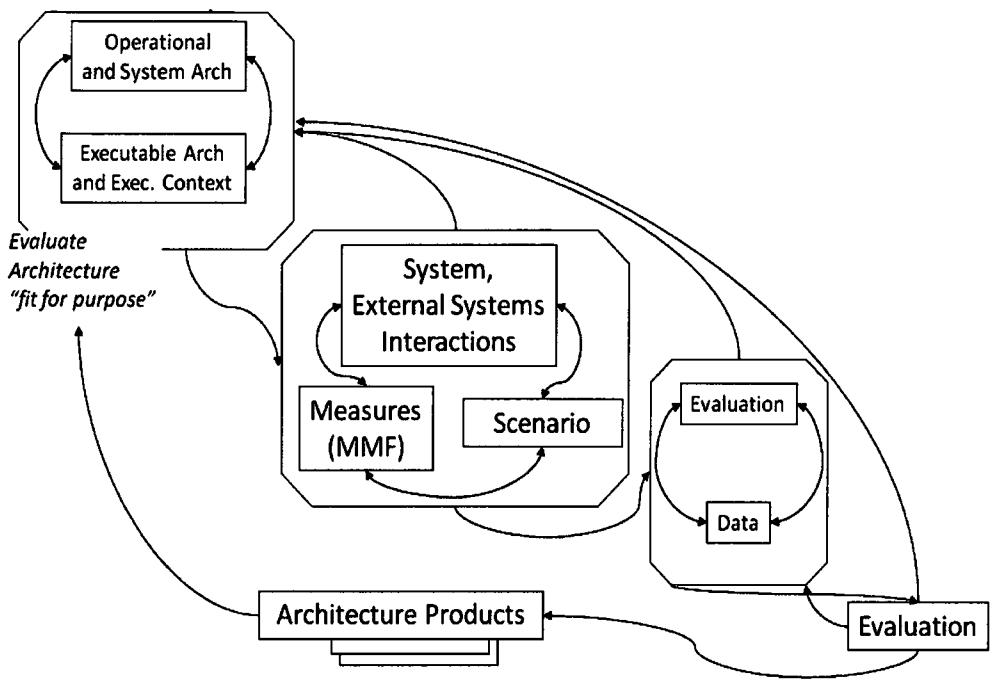


Figure 8: Executable context problem formulations

4.2. DoDAF AND EXECUTABLE ARCHITECTURES

DoDAF describes typical products as views prescribing high-level design artifacts, but leaves open the form in which the views are expressed. DoDAF products are textual, graphical and tabular items developed while building a description of known architecture elements and defining the characteristics relevant to the architectural purpose. The present DoDAF arranges products that represent static information on a variety of views. These static products may not be a reliable vehicle for detailed dynamic systems analysis and how these systems build interaction with each other.

Primarily, executable architecture descriptions are for analysis and must begin with an integrated, consistent, unambiguous architecture. DoDAF is the basis for the integrated architectures mandated in DoD Instruction 5000.2 and provides broad levels of specification related to operational, system and technical views (2003). Integrated

architectures are the foundation for interoperability in the Joint Capabilities Integration and Development System (JCIDS) prescribed in CJCSI 3170.01D and further described in CJCSI 6212.01D (2004; 2006). DoDAF and other DoD mandates pose significant challenges to the DoD system and operational architecture development since DoDAF specifications must be evaluated for compliance with requirements and objectives, yet they are not expressed in a form that is amenable to such evaluation. However, DoDAF compliant systems and operational architectures have the necessary information to construct high-fidelity simulations. Such simulations become, in effect, the executable architectures referred to in this research.

In this context, an integrated architecture is defined as a set of operational and systems architecture components which have similar sense, meaning, relationship, characteristics and properties. Among the multiple architectures, an integrated architecture can be defined while the similar, single architectures cannot, even if based on the identical set of DoDAF integrated products. United architecture elements can be rejoined for the next levels of development and analytical purposes. The program managers, domain experts and decision makers require these architectures to place, recognize and resolve definitions, facts, properties, constraints, issues and interfaces both across and within architectural boundaries. The impact and effect will be determined by the analysis.

4.2.1. EXECUTABLE ARCHITECTURE THAT APPLY DoDAF TECHNIQUES

Most studies regarding executable architectures are based on designing, evaluating and suggesting a path similar to other kinds of architecture modeling methods and techniques. The MITRE Corporation created the Executable Architecture Methodology for Analysis (EAMA) for analysis incorporating a combat model, communication model and process model to symbolize the main components of architecture and implementation of these models in the simulation environment. Joint Forces Command developed the Process Architecture and Analysis Model (PAAM), which is an operational and analytical tool used to inspect the effectiveness of future and current operational architectures (Pawlowski, et al., 2004; Garcia, & Browning,

2006). In reference to DoDAF, executable architecture, as it relates to the Method for Architecture Evaluations (MAVS), is used to assist in establishing the requirement for most information systems within the DoD (Garcia, 2010). One of the key DoDAF functions is to provide analysis worthy of military conduct. To provide this analysis, information-driven combat operations analysis leverages simulation technology to recognize the military value of Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance systems (C4ISR). This research investigates the usage of architecture descriptions based on the DoDAF to supply the required data for a dynamic-based model. It is enhanced through use cases from suggested operations center architectures. The conclusion from the literature reveals that the poor implementation of the DoDAF does not supply the necessary information for architecture evaluation.

In performing a comparison of the subject within the current literature, two points can be considered. First, it is difficult and complex to integrate products with DoDAF to produce executable architectures. Usually the philosophy of the integration methods is connected to the usage of systems and operational architecture models to produce executable architectures. The second point is how to leverage DoDAF in consistent approaches in producing executable architectures.

DoDAF is a widely-adopted architecture framework in the defense industry. DoDAF had its beginnings in the C4ISR community and is recognized as a basic part of the DoD's drive toward net-centric warfare.

4.2.2. INTEGRATION DEFINITION (IDEF)

Integration Definition (IDEF) is another modeling technique which can be utilized to enable knowledge-based architecture evaluation. IDEF was built by the US Air Force and it is presently being extended through knowledge-based organizations. Initially, it was developed to support the manufacturing industry. Methods of IDEF have been engaged for a wide range of uses, including the general development of software. IDEF's 16 methods from IDEF to IDEF14, including IDEFIX, are each intended to capture a similar kind of information by modeling procedures. IDEF methods are used to generate

graphical representations of multiple systems, examine the model and establish a model of a preferred version of the systems and assist the change from one to another. Occasionally, IDEF is used in connection with gap analysis.

The table below demonstrates the methods of IDEF that are either currently in existence or in developmental stages. The methods from IDEF0 to IDEF4 are most generally used.

IDEF METHODS	
IDEF0	Function Modeling
IDEF1	Information Modeling
IDEF1X	Data Modeling
IDEF2	Simulation Model Design
IDEF3	Process Description Capture
IDEF4	Object-Oriented Design
IDEF5	Ontology Description Capture
IDEF6	Design Rationale Capture
IDEF7	Information System Auditing
IDEF8	User Interface Modeling
IDEF9	Scenario-Driven IS Design
IDEF10	Implementation Architecture Modeling
IDEF11	Information Artifact Modeling
IDEF12	Organization Modeling
IDEF13	Three Schema Mapping Design
IDEF14	Network Design

Table 3: IDEF methods

As an illustration of the procedures, the methods of IDEF0 are intended to model the purpose of an enterprise, generating a graphical model which indicates what directs

the function, who executes, what resources are carried out in its use, what the factors of production are and what dealings and relationships it has with other utilities. In other words, the IDEF0 aids in answering the information interrogatives of the research (who, what and where) of the context of the systems.

IDEF3 was created to assist systems modeling within the business world by capturing explanations of activities lists. For a specific scenario, an IDEF3 diagram may indicate the sequence of procedures, which procedures occur in a similar fashion, where choices exist, points, etc., making the IDEF3 into more of a diagram. The IDEF3 can be an influential tool to recognize the performance and functions of a systems architecture and is leveraged in this research. Figure 9 below is an example IDEF3 diagram using the IDEF3 or UOB symbols in Figure 10. The example indicates a decision point following a procedure marked "Evaluate Proposal." In the case that the decision is positive, the next connection or junction demonstrates that any path will result in the last procedure "Award Contract." The diagram of IDEF3 demonstrates a proper sequence of procedures.

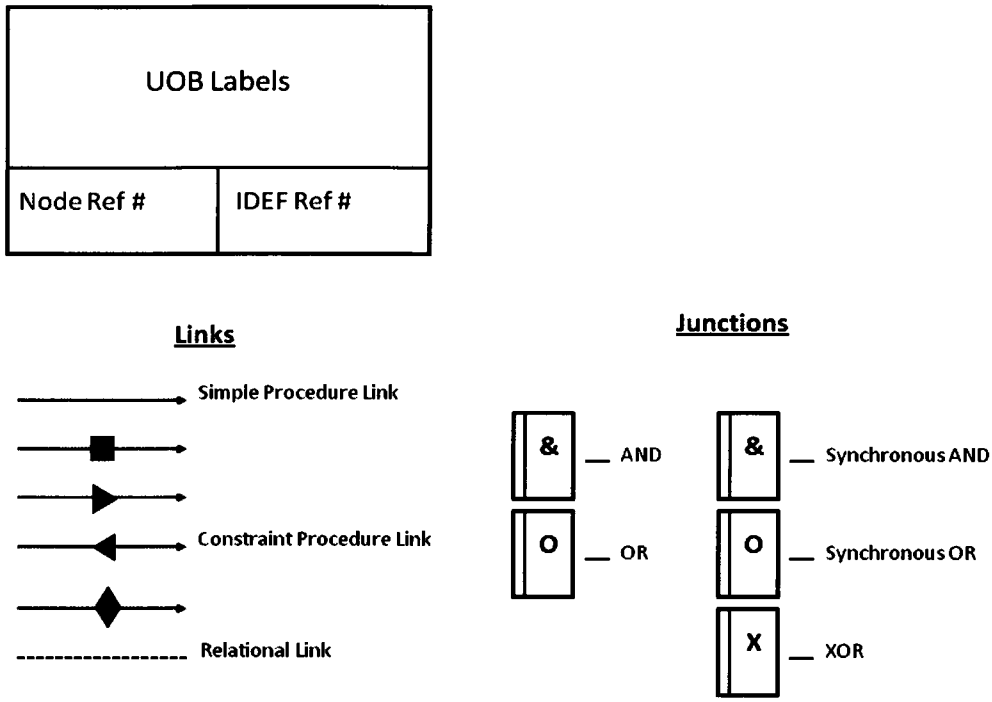


Figure 9: IDEF 3 symbols (UOB symbols)

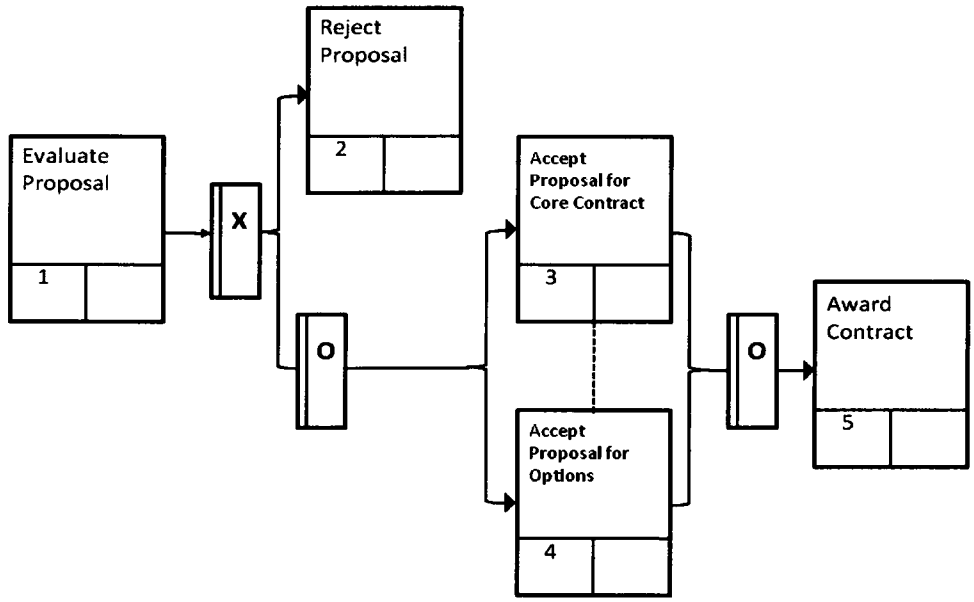


Figure 10: Examples of IDEF 3 diagram

These procedures in IDEF modeling can assist in gathering the needed information to answer the "when" of the information interrogatives. On its own, IDEF is not as advanced as the executable architectures discussed in the previous section, but is leveraged as part of the EC research to enable data gathering of non-DoDAF elements and filling data gaps to gather metrics and measures.

4.3. ZACHMAN

The business society frequently utilizes the Zachman framework, which was created in the 1980s by John Zachman. From Zachman's viewpoint, the framework was created to help companies deal with the dynamics and complexities of the information age. The framework is fundamentally a matrix of 36 cells which represent the *how* (function), *what* (data), *who* (people), *where* (network), *why* (motivation) and *when* (time) at six deferent stages from prospective to detail and is the motivation behind the development of DoDAF.

While the Zachman framework, along with other frameworks, was established in the world of business, the DoD required something customized to its needs. Most of the frameworks were created to promote and sell services and goods, uses that are unrelated to DoD. The Architecture Working Group (AWG) released C4ISR Architectural Framework Version 1.0 in 1996. Within a year, AWG implemented much required revisions and additions to the C4ISR Architectural Framework, and Version 2.0 was released. According to the *Under Secretary of Defense (USD) 23 Feb 1998 Memorandum* cited by Andrew W. Zinn, it was stated that, "We see the C4ISR Architecture Framework as a critical element of the strategic direction in the Department, and accordingly direct that all ongoing and planned C4ISR or related architectures be developed in accordance with Version 2.0."

As mentioned earlier in the introduction, Zachman uses six interrogatives: *who*, *what*, *where*, *when*, *why* and *how* (Zachman, 1997). To illustrate the function of the EC concept, these six interrogatives are further broken down to illustrate EC as a method

that enables information (*who*, *what*, *where*), enriches executable architectures (*when*) and enhances the creation of knowledge (*why*, *how*). Table 4 shows the Zachman Enterprise Architecture Framework and expands upon his use of *what*, *how*, *where*, *who*, *when* and *why*.

	DATA *What*	FUNCTION *How*	NETWORK *Where*	PEOPLE *Who*	TIME *When*	MOTIVATION *Why*
Objective/Scope (contextual) *Role: Planner*	List of things important in the business	List of Business Processes	List of Business Locations	List of important Organizations	List of Events	List of Business Goal & Strategies
Enterprise Model (conceptual) *Role: Owner*	Conceptual Data/ Object Model	Business Process Model	Business Logistics System	Work Flow Model	Master Schedule	Business Plan
System Model (logical) *Role:Designer*	Logical Data Model	System Architecture Model	Distributed Systems Architecture	Human Interface Architecture	Processing Structure	Business Rule Model
Technology Model (physical) *Role:Builder*	Physical Data/Class Model	Technology Design Model	Technology Architecture	Presentation Architecture	Control Structure	Rule Design
Detailed Reprentation (out of context) *Role: Programmer*	Data Definition	Program	Network Architecture	Security Architecture	Timing Definition	Rule Speculation
Functioning Enterprise *Role: User*	Usable Data	Working Function	Usable Network	Functioning Organization	Implemented Schedule	Working Strategy

Table 4: Zachman enterprise architecture framework interrogatives

4.4. 4+1 VIEW MODEL

There are numerous ways to view or build architecture models. One such model is the 4+1 View Model. The 4+1 View Model describes software architecture using five concurrent views, each of which addresses a specific set of concerns. The logical view describes the design's object model; the process view describes the design's concurrency and synchronization aspects; the physical view describes the mapping of

the software onto the hardware and shows the system's distributed aspects; and the development view describes the software's static organization in the development environment.

According to Kruchten, by using five synchronized views, the 4+1 model illustrates the architecture of software (1995). These views include the logical view, development view, process view, physical view and scenario view. Every view deals with a definite set of concerns. The object model of design, the services to be provided by the system to users, is described by the logical view. The non-functional features of synchronization and concurrency of the design is described by the process view. The concentration on actual software, the statistic management of the software in the environment of development, is illustrated by the development view. The software mapping against the hardware is described by the physical view and also shows the distributed features of the system. The software designers can manage the description of their architectural decisions around these four views and demonstrate them with some preferred scenarios or use cases that represent a fifth view. These views are related to knowledge evaluation of a system-of-systems and provided some context for the EC methods development.

The architecture is developed partly by using these scenarios. The different stakeholders could find their requirement in the software architecture through the 4+1 model. Through the physical view, the system engineers approach this 4+1 view model first, then via the process view. Through the logical view, customers, data specialists and end users can view the architecture. Staff members of software configuration and project managers use the model through the development view (Kruchten, 1995).

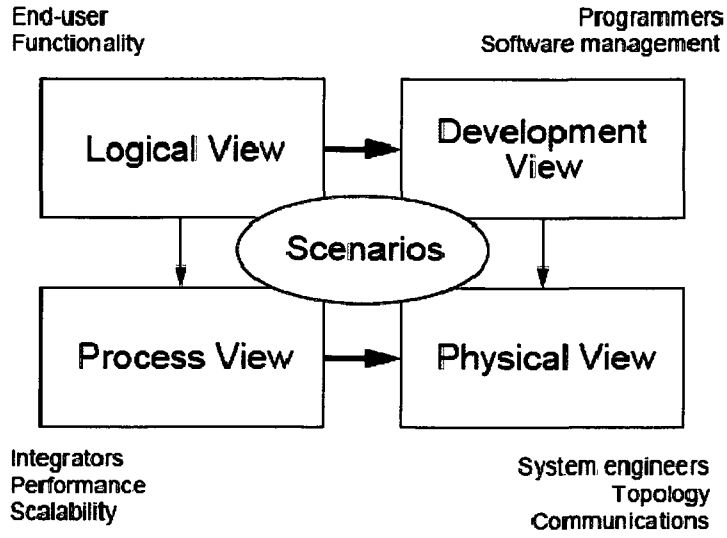


Figure 11: The 4+1 view model

4.4.1. LOGICAL ARCHITECTURE

The logical architecture primarily supports the functional system needs of the users in the service terms. The system is deducted to a key abstraction set consumed by the problem domain in the object class' format, which develops the principles of inheritance, abstraction and encapsulation. In addition to functional analysis, it also provides the ability to recognize design elements and common mechanisms over the different divisions of the system.

4.4.2. LOGICAL VIEW NOTATION

The logical view notation is derived from the notation of Booch Object-Oriented Design object modeling language and methodology that was widely used in object-oriented analysis and design. The notation aspect of the Booch method has now been superseded by the Unified Modeling Language (UML), which features graphical elements from the Booch method along with elements from the object-modeling technique (OMT) and object-oriented software engineering (OOSE). It is streamlined to consider only the architecturally important items.

4.4.3. PROCESS ARCHITECTURE

Process architecture considers few non-functional necessities such as availability and performance. It deals with the distribution and concurrency of a system's reliability, fault tolerance and the adjustment procedure of major logical view abstractions within the process architecture. The process architecture can be illustrated at different stages of the abstraction, as each stage deals with various concerns. At the major level, the process architecture can be considered as a set of logical networks communicating with independently executed programs, circulated over a set of hardware resources joined by a WAN or LAN. Distributing the same physical resources, the multiple logical networks may exist concurrently. A process is a combination of executable unit tasks. The processes indicate the stage of deliberately-controlled process architectures. Also, for the improved allocation of the processing load, the processes can be simulated (Kruchten, 1995).

4.4.4. DEVELOPMENT ARCHITECTURE

In the software development environment, architecture development concentrates on the authentic software organization module. The software is enclosed in small portions called the subsystems or the program libraries, which can be developed by few developers. In a hierarchy of layers, the subsystems are maintained, supplying a distinct and narrow interface by each layer to the above layers. Through the subsystem and module diagrams, the system's development architecture is indicated, displaying the associations of exports and imports. Only after recognition of all the software elements can the total development architecture be illustrated.

Most development architectures consider the internal needs associated with the ease of development, reuse, software management and the programming language or toolset limitations. The development outlook is the source for the requirement allocation of team tasks or team organization; cost assessment and planning; observation of project improvement; and software reusability, security and portability

analysis. To establish the line of product, the development view is the foundation (Kruchten, 1995).

4.4.5. PHYSICAL ARCHITECTURE

The physical architecture mainly considers the system's non-functional needs such as performance, reliability, scalability and availability. The software will run on the computer networks or on the processing nodes. The objects, processes and tasks are the different recognized elements which must be mapped onto the different nodes. Many varied physical configurations should be used, including a few for testing and development and the remainder for system deployment for different sites and clients. Consequently, the node software mapping requires increased flexibility and less impact on the source code (Kruchten, 1995).

4.4.6. SCENARIOS

The elements of the four views are intended to work collectively by the small, significant set of scenarios. The scenarios are an abstraction of the major significant necessities. The design is articulated by using object scenarios, pictures and object interaction pictures. It supports two primary objectives: determine the architectural elements when designing the architecture and illustrate and evaluate tasks on the completion of architecture design. Both of these tasks are performed as the starting point of an architectural prototype test (Kruchten, 1995).

The EC methodology requires the use of all of these views in order to produce a evaluated architecture. Content can be added to each view to aid in the results of the overall methodology.

4.5. MODEL-DRIVEN ENGINEERING

The process of model-based software engineering is generally addressed as Model-Driven Engineering (MDE). This method improves the model before the end product or artifact is designed, and following the design of the end product or artifact, the model is renovated to reflect the actual artifact. Model-driven engineering (MDE) is

a software development methodology which focuses on creating and exploiting domain models (abstract representations of the knowledge and activities that govern a particular application domain), rather than on the computing (or algorithmic) concepts. The MDE approach is meant to increase productivity by maximizing compatibility between systems (via reuse of standardized models), simplifying the process of design (via models of recurring design patterns in the application domain), and promoting communication between individuals and teams working on the system (via a standardization of the terminology and the *best practices* used in the application domain).

A modeling paradigm for MDE is considered effective if its models make sense from the point of view of a user that is familiar with the domain and can serve as a basis for implementing systems. The models are developed through extensive communication among product managers, designers, developers and users of the application domain. As the models approach completion, they enable the development of software and systems.

Some of the better known MDE initiatives are:

- The Object Management Group (OMG) initiative Model-Driven Architecture (MDA), which is a registered trademark of OMG (Object Management Group, 2009).
- The Eclipse ecosystem of programming and modeling tools.

An MDE tool is utilized to interpret, compare, develop and align models and meta-models. More than one tool may control all of the features required for MDE. The UML utilized in MDE is a minute subset of great broader range of UML. As a division of MDE, the UML is enclosed by its own UML meta-model. Development has been made to progress models of executable UML, even though it has not received industry majority acceptance when used for the same limited range (Object Management Group, 2009).

MDE encourages efficient use of system models in the development process and it supports reuse of best practices when creating system-of-systems (Brown, 2009). According to Douglas C. Schmidt, model-driven engineering technologies offer a

promising approach to address the inability of third-generation languages to alleviate the complexity of platforms and express domain concepts effectively (Schmidt, D.C, 2006).

4.6. MISSIONS AND MEANS FRAMEWORK (MMF)

The MMF is the method used to provide military mission specifications and to qualitatively estimate the mission's effective use of alternative war fighting Doctrine, Organization, Training, Materiel, Leadership, Personnel, and Facilities (DOTMLPF) services. The MMF was developed by Deitz, et al. and enables architectures representation to specify the military mission and, therefore, quantitatively evaluates the mission utility of alternative warfighting DOTMLPF services and products (Dietz, et al., 2004). This research leveraged mapping of the MMF entities to DoDAF views.

This mapping is an essential piece that enables the EC to develop the much needed measures against the EC problem formulation. The MMF first amalgamates top-down and then merges bottom-up as illustrated by Figure 12 below. The MMF segments were used as necessities in the development and testing for the Army's planned Future Combat Systems - equipped Unit of Action.

A Measure of Performance (MOP) is a criterion used to assess friendly actions that are tied to measuring task accomplishment [JP 1-02, Appendix A-1, p.333, 3/2007]. Measure of Effectiveness is a criterion used to assess changes in system behavior, capability, or operational environment that is tied to measuring the attainment of an end state, achievement of an objective, or creation of an effect [JP 1-02, Appendix A-1, p.333, 3/2007]. MOP and MOE will be described in greater detail in the results sections of Chapter 6.

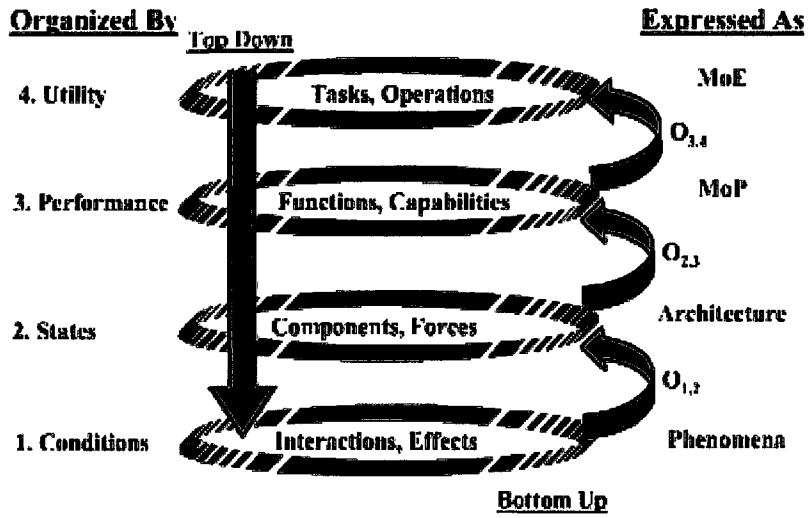


Figure 12: The synthesis and employment processes for the "how and why"

4.6.1. MAPPING TO DoDAF

The following section describes the detailed experiments conducted to support the evaluation of this research in an operational environment. Table 5 represents how MMF operators are mapped to DoDAF views. This mapping enables EC to represent the means required to support architectures evaluation. The MMF provides a way to describe military operations domain using the language of military science in a manner that can be digested and used by those supporting the warfighters and also be readily presented back to and understood by the warfighters. The MMF provides a structured way to describe key elements of military operations that are essential to understand in order to successfully model and simulate those operations. The framework provides the necessary structure to support a disciplined, repeatable procedure to explicitly specify the mission and assess mission accomplishment. Used in conjunction with automated knowledge acquisition and integration tools, the framework supports the operator's ability to capture the products of key portions of the top – down planning and decision

making process in data element form rather than just text and graphics, whether manually generated or machine generated. Because tasks, the building blocks of missions, are pulled from authoritative sources, common and commonly accepted terms and definitions are built into the framework methodology. Components, which represent the means used to execute tasks, are similarly derived from authoritative sources and other databases.

Conditions and standards for specific tasks are established based on the results of mission analysis, Course of Action (COA) development and war gaming during the planning and decision making process. The same task may be iterated several times with different sets of conditions and standards based on when and where the task iteration is to occur within the concept of operations. Measures and criterion used to develop standards may be structured to provide quantitative metrics in the form of Measures of Performance (MoP), which describe minimum acceptable levels of performance in terms of time, distance, accuracy, etc. Standards may also be structured to provide more qualitative metrics in the form of Measures of Effectiveness (MoE), which describe the desired end state or purpose of the task. MoPs are also extremely useful in an operational context in defining the level of performance required under a given set of conditions to enable the entity performing the task to accomplish the purpose (MoE) of that task or to enable a different (higher, lower, adjacent) entity to accomplish the purpose of a related task. Consequently, it is possible to establish a link between required performance (MoP) and desired effect (MoE) within the context of an operation.

Level 7: Purpose, Mission	•OV-1, AV-1 •The "why" and "wherefore." An assignment with a purpose that indicates the action to be taken. "What" the required outcomes are and "who" has been assigned them.
Level 6: Environment Context	• AV-1 •"Under what circumstances" a mission is to be accomplished.
Level 5: Index, Location/Time	• OV-1, AV-1 • "Where" (geo-spatial) and "when" with what TPFDD execution matrix.
Level 4: Tasks, Operations	• OV-5 • Task-based, outcome-centric specification of operations that provide the means to accomplish the mission. Objective: organize task outcomes and evaluate mission effectiveness.
Level 3: Functions, Capabilities	• OV-5, SV-11 • Function-based, performance-centric "how well" specifications of capabilities.
Level 2: Components, Forces	• OV-2, OV-3, OV-4, All SV • Component-based, state-centric specifications of the forces that provide the means. Network of units, personnel, and equipment. Physical and logical networking.
Level 1: Interactions, Effects	• OV-6a, OV-6b, OV-6c, OV-7, SV-10a, SV-10b, SV-10c • Interaction-based, phenomena-centric specification of effects of operations on forces.

Table 5: MMF mapping to DoDAF

5. METHOD DEVELOPMENT OVERVIEW

This section gives an overview of how the identified methods are applied in detail to provide the support needed to address the earlier identified research questions and how they are combined into the proposed framework ECSF.

5.1. DISCRETE EVENT SYSTEM SPECIFICATION (DEVS) AND DEVS UNIFIED PROCESS (DUNIP)

The Discrete Event System Specification (DEVS) formalism is a general enough approach to handle the complex hierarchical nature of architectures and the interrelationships of the views and elements. Saurabh Mittal, of DUNIP Technologies, postulates and identifies a shortcoming or oversight in the DoDAF standard adopted by the DoD. Mittal suggests that, "DoDAF doesn't mandate any simulation methodology to analyze the system or perform any pre-design feasibility studies" (Mittal, et al. 2007). In summary, DoDAF does not lend itself to the M&S field, even though M&S would be an invaluable tool to evaluate DoDAF specifications to verify that requirements and objectives are met.

Executable context, therefore, extends DUNIP to create an executable federate that allows for the use of DEVS JAVA properties that contains the meta-data that helps convert the models into a common federate and provides for operational and systems model transparency. This creates operational and system model harmonization for resulting artifacts to be federated into an executable context, as identified in the literature research, than what has historically been done.

DEVS Unified Process (DUNIP) uses the DEVS formalism as a basis for automated generation of models from various requirement specifications and realization as collaborative services (Mittal, 2006). See Figure 13.

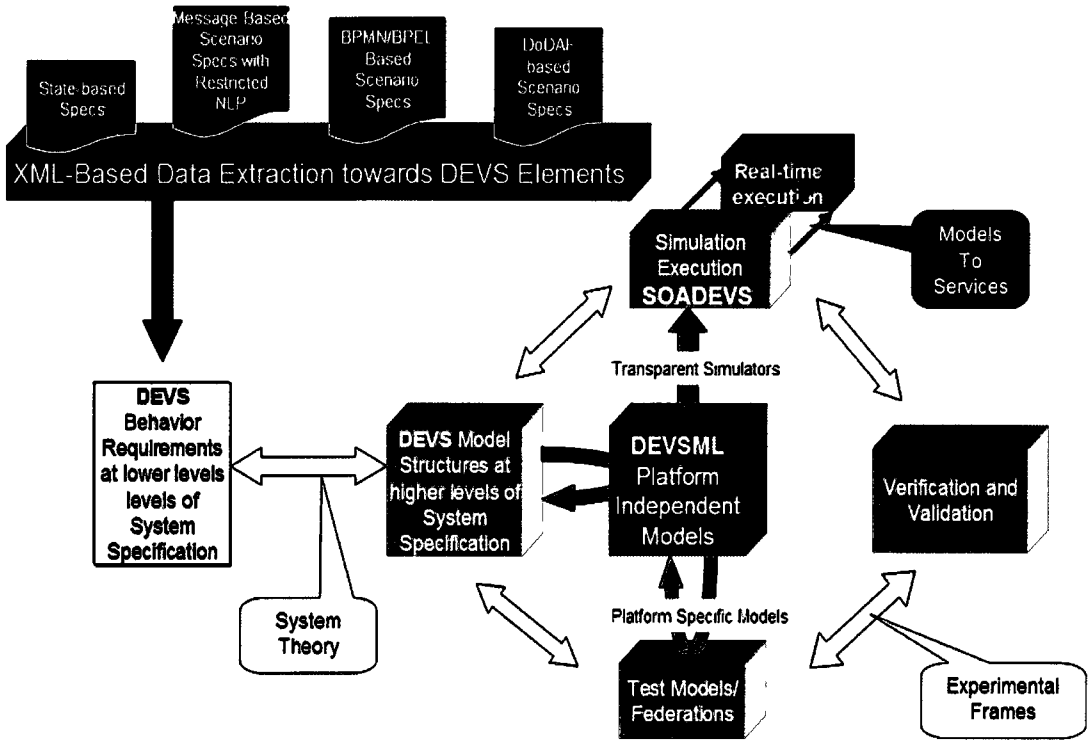


Figure 13: DEVS Unified Process (DUNIP)

This research utilized DUNIP to express its architecture models to establish a valid model leveraging Figure 13 in order to develop the following steps:

1. Develop the requirement specifications in DoDAF.
2. Use the DEVS-based automated model generation process to generate the DEVS atomic and coupled models from the requirement specifications using XML
3. Evaluate the generated models using DEVS W3C atomic and coupled schemas to make them capable for collaborative development,
4. From Step 2, simulate the coupled model using DEVS
5. Execute the simulation on an isolated machine or in distributed manner using SOA. Execute the simulation in real-time as well as in logical time.
6. The test-suite generated from DEVS models can be executed in the same manner as laid out in Step 2.

7. Compare the results from Step 5 and Step 6 to evaluate the architecture.

DEVS is inherently based on object-oriented methodology and systems theory and categorically separates the model, the simulator, and the experimental frame; it has been used to model systems over the years. Mittal also proposed a mapping of DoDAF architectures into a computational environment that incorporates dynamical systems theory and an M&S framework (Mittal, 2006).

Zeigler developed the DEVS formalism that supports systems engineering of discrete events in a modular and hierarchical method. The formalism provides a mathematical basis for studying discrete event systems for good understanding. It has been used largely for replication and modeling because of its mathematical foundation (Zeigler, 2003). The research activities associated with the DEVS theory have been developed in three directions in the past years: theory, methodology and applications. The DEVS formalism's applicability to performance measurement, logical analysis, and discrete event control has been confirmed through expansion of formalism and adaptation of the other theories (Zeigler, 2003).

The methodology will support complex information systems specification and evaluation using advanced simulation capabilities. Specifically, the DEVS formalism will provide the basis for the computational environment with the systems theory and M&S attributes necessary for design modeling and evaluation. DUNIP demonstrated how this information is added and harnessed from the available DoDAF products toward development of an extended DoDAF integrated architecture that is "executable." This research focused on adding minimal information to enable DoDAF to become the executable architecture for the knowledge-based method developed.

5.2. FEDERATING EXECUTABLE CONTEXT WITH FEDEP/DSEEP

IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP) is a standard developed by the Simulation Interoperability Standards Organization (SISO) (IEEE Std. 1730-2010). The standard outlines recommended high-level processes that should be adopted throughout the

development lifecycle of distributed simulations. It has much in common with the systems engineering lifecycle and provides additional guidance to an organization's standard processes, specifically tailored to the needs of personnel involved in producing M&S environments. DSEEP is a generalized evolution of the IEEE 1516.3 Federation Development and Execution Process (FEDEP) that has similar aims but is tailored explicitly toward distributed simulations. The DSEEP IEEE standard provides recommended practice of the Distributed Simulation Engineering and Execution Process (DSEEP). The DSEEP is intended as a high-level process framework into which the lower-level systems engineering practices native to any distributed simulation user can be easily integrated. Simulation architectures include Distributed Interactive Simulation (DIS), High Level Architecture (HLA), and Test and Training Enabling Architecture (TENA).

The DSEEP is comprised of seven steps that define the entire lifecycle of an M&S application from initial concept to results analysis. Each step is divided into activities. This process is explained in Table 6.

Step 1: Define Simulation Environment Objectives	•**Activity 1.1.** Identify User and Sponsor Needs •**Activity 1.2.** Develop Objectives •**Activity 1.3.** Conduct Initial Planning
Step 2: Perform Conceptual Analysis	•**Activity 2.1.** Develop Scenario •**Activity 2.2.** Develop Conceptual Model •**Activity 2.3.** Develop Simulation Environment Requirements
Step 3: Design Simulation Environment	◦ **Activity 3.1.** Select Members • **Activity 3.2.** Prepare Simulation Environment Design • **Activity 3.3.** Prepare Detailed Plan
Step 4: Develop Simulation Environment	• **Activity 4.1.** Develop Simulation Data Exchange Model • **Activity 4.2.** Establish Simulation Environment Agreements • **Activity 4.3.** Implement Member Application Designs • **Activity 4.4.** Implement Simulation Environment Infrastructure
Step 5: Plan, Integrate and Test Simulation Environment	• **Activity 5.1.** Plan Execution • **Activity 5.2.** Integrate Simulation Environment • **Activity 5.3.** Test Simulation Environment
Step 6: Execute Simulation Environment and Prepare Outputs	• **Activity 6.1.** Execute Simulation • **Activity 6.2.** Prepare Simulation Environment Outputs
Step 7: Analyze Data and Evaluate Results	• **Activity 7.1.** Analyze Data • **Activity 7.2.** Evaluate and Feedback Results

Table 6: DSEEP seven-step process

Within this research, DSEEP and FEDEP were used as a key step in the method. After the DoDAF model had been extended to include provisions for M&S through DEVS, the features of DSEEP and FEDEP were applied to the new architecture to aid in decision-making, reducing risk, training system selection, and test and evaluation. Specifically, DSEEP can be applied to determine the right mix of systems to employ in the architecture, thereby reducing the risk of using systems that may not function best for the particular mission. Furthermore, results of the DSEEP process can be used to determine which systems to test and evaluate further for inclusion or exclusion in the target architecture. DSEEP can prove to be a valuable part of the EC methodology by narrowing the decision making process, minimizing risks and highlighting what is suitable for further testing. The DSEEP has been designed to serve as the generalized framework from which alternative and more detailed views can be specified in order to better serve the specialized needs of specific communities. Such views provide more detailed "hands-on" guidance to users of this process from the perspective of a particular domain (e.g., analysis, training), a particular discipline (e.g., VV&A, security), or a particular implementation strategy (e.g., HLA, DIS, TENA).

5.3. MEASURES OF EFFECTIVENESS (MOE) AND MEASURES OF PERFORMANCE (MOP)

This experimental example provides data that has been expanded in the research to show that the EC method enables architecture evaluation in gathering: Measures of Effectiveness (MoE) and Measures of Performance (MOP). Traditional measures of effectiveness (MOE) and measures of performance (MOP) practice have focused on forces-based, material-centric measures such as time required completing an operation. The MMF was used to focus on Mission-centric Measures within EC. Here, MoE and MoP measures and standards are the codification of how planned/delivered task outcome affects Mission success. In many cases, the required task involves a specific system and a desired condition that enables the use of the system; EC enables the use of the MMF and the federation of the operational and systems artifacts to ensure that all measures of performance on the system level are computed within the

user relevant operational context based on the system's specification and contribute directly to the operational efficiency (MOE) and systems efficiency (MOP). Furthermore, the resulting level of detail allows for specific evaluation of system interactions in the context of being part of the operationally specified system-of-systems, so that detailed analysis of system behavior in the operational context becomes observable.

6. EXPECTED RESULTS

This section provides two examples of how a knowledge-based approach enables evaluation of theoretical and operational conditions.

The first example is a well-documented problem in the field of computer science and computer engineering: deadlock and livelock systems. This example requires the detailed modeling of interactions between the specified system and other systems in the operational context. As current solutions on the operational level do not use detailed specification in the form of architecture to model their system, such observations are not supported by this category. As current solutions for executable architectures do focus on the system and do not take the operational context sufficiently into account, this second category of current solution is no alternative to the proposed framework as well. If the experiment with the implemented framework shows such an example, the contribution of this research to close part of the gap is made.

The second example is evaluating the contributions of a new system added to an existing operational process within the DoD and compares it with a current solution. This will provide insight into how metrics can be gathered to evaluate how the new system operates within its operational context. It requires the consistent application of system level measures of performance for the system providing the current solution as well as for the new system that provides the alternative solutions. As the same methods are applied to define the current and new system based on its specification, the comparison of their operational contribution is based on equal and comparable engineering specifications, and not on the assumptions of model developers. As such comparisons are not feasible with current approaches; the demonstration of the feasibility of such an experiment is an innovative contribution.

6.1. FIRST EXAMPLE: DEADLOCK

Because EC can be generally applied, there are multiple ways it can solve the deadlock problem. In computer science, deadlock refers to a specific condition when two or more processes are each waiting for each other to release a resource, or more

than two processes are waiting for resources in a circular chain (Mogul, et. al., 1996; Anderson, et. al., 2001; Zobel, D., 1983). There are four general properties that must hold to produce a deadlock.

6.1.1. MUTUAL EXCLUSION

When one thread owns some resource, another cannot acquire it. This is the case with most critical sections, but is also the case with GUIs in Windows. Each window is owned by a single thread, which is solely responsible for processing incoming messages; failure to do so leads to lost responsiveness at best and deadlock in the extreme.

6.1.2. A THREAD HOLDING A RESOURCE IS ABLE TO PERFORM AN UNBOUNDED WAIT

For example, when a thread has entered a critical section, code is ordinarily free to attempt acquisition of additional critical sections while it is held. This typically results in blocking if the target critical section is already held by another thread.

6.1.3. RESOURCES CANNOT BE FORCIBLY TAKEN AWAY FROM THEIR CURRENT OWNERS

In some situations, it is possible to steal resources when contention is noticed, such as in complex database management systems (DBMSs). This is generally not the case for the locking primitives available to manage code on the Windows platform.

6.1.4. A CIRCULAR WAIT CONDITION

A circular wait occurs if a chain of two or more threads is waiting for a resource held by the next member in the chain. Note that for non-reentrant locks, a single thread can cause a deadlock with itself. Most locks are reentrant, eliminating this possibility.

Deadlock can be modeled with a directed graph. In a deadlock graph, vertices represent either processes (circles) or resources (squares). A process which has acquired a resource is shown with an arrow (edge) from the resource to the process. A process which has requested a resource that has not yet been assigned to it is modeled with an

arrow from the process to the resource. If these create a cycle, there is deadlock. The deadlock situation described above can be modeled like this:

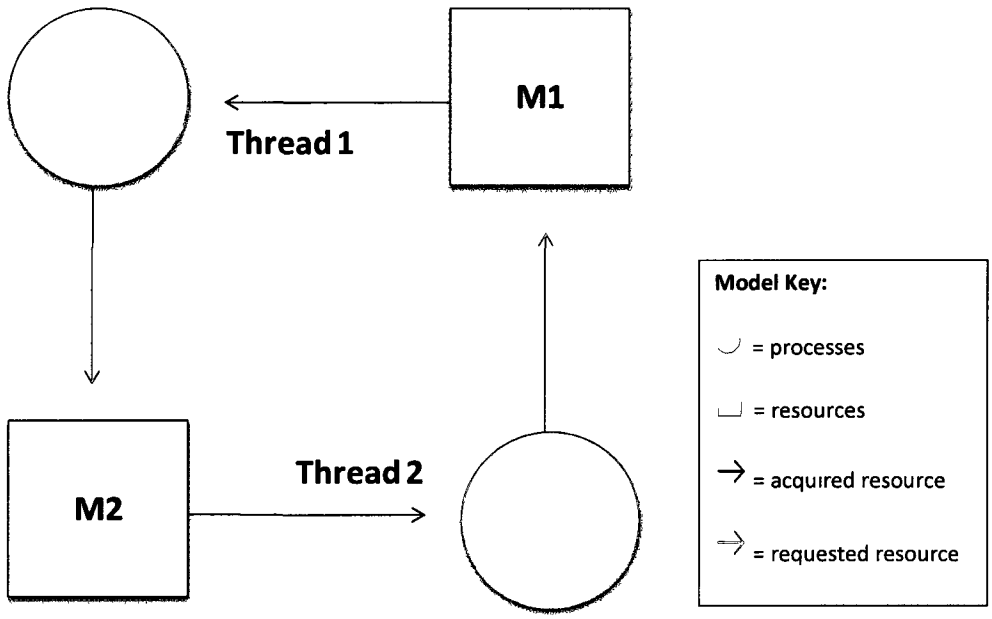


Figure 14: Deadlock model

The deadlock model shown above illustrates an extremely simple deadlock situation, but it is also possible for a more complex situation to create deadlock. The following is an example of deadlock with four processes and four resources:

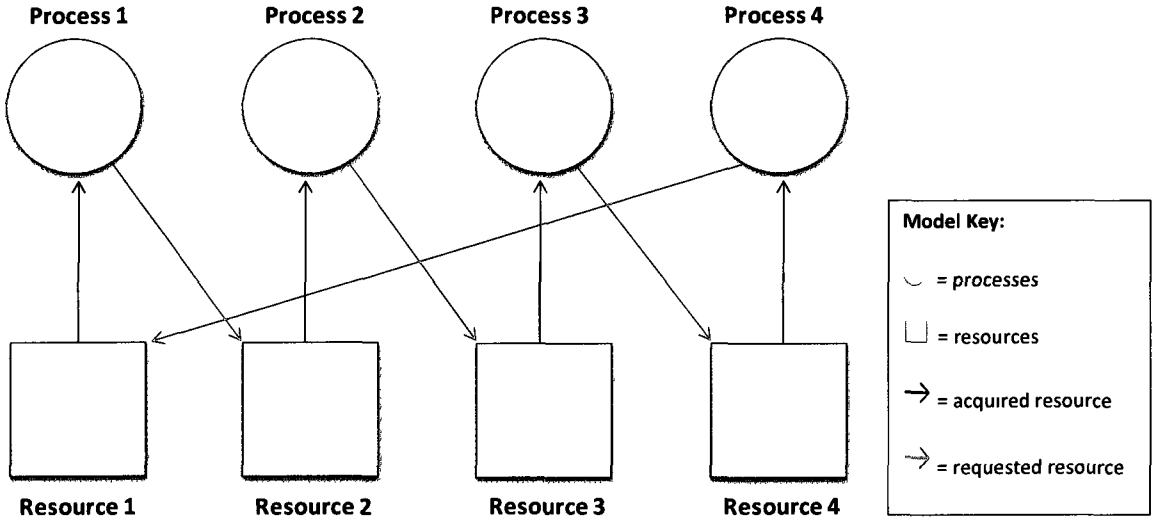


Figure 15: Deadlock with four processes and four resources

6.1.5. SUMMARY OF DEADLOCK

Deadlock is a set of processes in which each process in the set is waiting for an event that only another process in the set can cause (Sirer, 2001; Rensselaer, D.H., 2004; Venkatesh, J., et al., 2000). The event is usually the release of a currently-held resource. As a result, none of the processes can run, release resources or be awakened.

6.2. THEORETICAL EXAMPLE OF EC: LIVELOCK

A livelock is similar to a deadlock, except that the state of the two processes involved in the livelock constantly changes with regards to the other process. It occurs when a process repeats itself because it continues to receive erroneous information. It can also occur when a process calls another process and is then called by that process with no logic to detect this situation and stop the operation. A livelock differs from a deadlock in that processing continues to take place, rather than just waiting in an idle loop. As a real world example, livelock occurs when two people meet in a narrow corridor, and each tries to be polite by moving aside to let the other pass, but they end up swaying from side to side without making any progress, moving the same way at the same time. In general, the term usually connotes one of the following:

6.2.1. STARVATION

Systems with a non-zero service cost and unbounded input rate may experience starvation. For example, if an operating system kernel spends all of its time servicing interrupts, user processes will starve.

6.2.2. INFINITE EXECUTION

The individual processes of an application may run successfully, but the application as a whole may be stuck in a loop. For example, a naive browser loads web page "a" that redirects to page "b" which erroneously redirects back to page "a". Another example is a process stuck traversing a loop in a corrupted linked list.

6.2.3. BREACH OF SAFETY PROPERTIES

The safety property of distributed systems states that the application will not perform an incorrect action or enter an undesirable state. By adding a temporal attribute to the application state, the program is considered live locked if it does not make forward progress within a specified timeframe. For example, if the temporal rule that a response is sent for every request within 10 seconds fails, then the server is deemed to be at a standstill. Creating the appropriate specifications for a given application requires detailed domain knowledge about the program's intended behavior and internals of its implementation. In summary, livelock is a situation in which a block returns to the same state infinitely, often at the same instant.

6.2.4. ANALYTICAL ALGORITHMS

A pessimistic algorithm detects contention when attempting to acquire a shared resource, usually responding by waiting until it becomes available (for example, blocking). Optimistic algorithms attempt forward progress with the risk that contention will be detected later on, such as when a transaction attempts to commit.

Lock-free or interlocked-based algorithms that can detect and respond to contention are relatively common for systems-level software; these algorithms often avoid entering a critical section in the fast path, choosing to deal with livelock instead of

deadlock. Livelock presents a challenge to parallel code and is caused by fine-grained contention. The result stalls forward progress much like a deadlock.

In the example below, threads 1, 2 and 3, and three locks A, B and C are involved in some form of shared-memory coordination. Thread 1 holds lock A and is blocked on acquiring lock B; thread 2 holds lock B and is blocked on acquiring lock C; thread 3 holds lock C. If thread 3 then attempts to acquire lock A, the algorithm initiates and constructs a wait graph like that depicted in Figure 16. Then it will detect a cycle and respond by terminating thread 3. This frees up lock C, which enables thread 2 to unblock, acquire C, execute and release B. This unblocks thread 1, which is then able to acquire B and execute to completion.

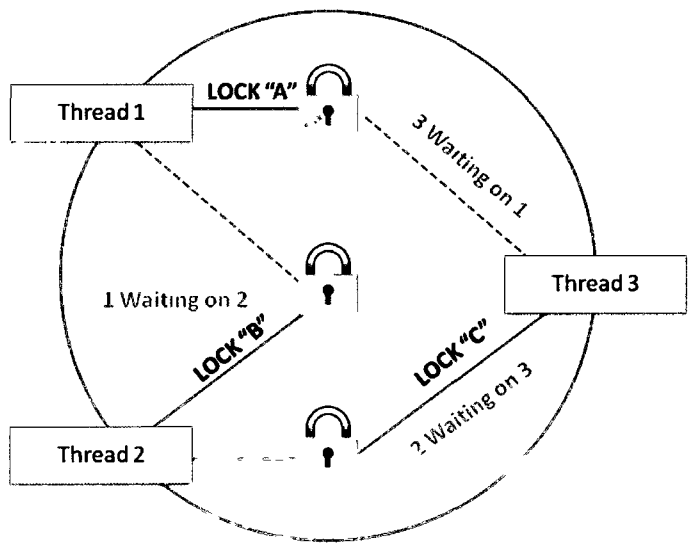


Figure 16: Wait graph – deadlock situation with termination of thread 3 avoidance

6.2.5. DEADLOCK AVOIDANCE

Deadlocks can be avoided if certain information about operational processes is available in advance of resource allocation. For every resource request, the system sees if granting the request will enter the system into an unsafe state, one that could result

in deadlock. The system then only grants the request that will lead to safe states. In order for the system to determine if the next state will be safe or unsafe, it must always share advance knowledge of the number and type of all resources in existence, available and requested.

To avoid livelock and related problems, an operating system must schedule a network interrupt as carefully as it schedules process execution. Furthermore, using a modified interrupt-driven networking implementation, this will eliminate livelock without degrading other aspects of system performance.

6.2.6. DEADLOCK/LIVELOCK RECOVERY

Once a deadlock is detected, there are two choices:

- Abort all deadlocked processes (which will cause some computations to be repeated)
- Abort one process at a time until cycle is eliminated (which requires re-running the detection algorithm after each abort)

A further consideration is process preemption. Process preemption releases resources until the system can continue. However, process preemption involves certain issues including:

1. Selecting the victim
2. Rollback
3. Programming model
4. Starvation
5. Livelock

Livelock is a risk with some algorithms that detect and recover from deadlock. If more than one process takes action, the deadlock detection algorithm can repeatedly trigger. This can be avoided by ensuring that only one process (chosen randomly or by priority) takes action.

6.2.7. DEADLOCK/LIVELOCK PREVENTION

Simply put, deadlock/livelock can be prevented by ensuring that one of the above five conditions does not occur. Further, removing the mutual exclusion condition means that no processes have exclusive access to a resource. This proves impossible for resources that cannot be spooled, and even with spooled resources, deadlock could still occur.

Hold and wait conditions may be removed by requiring processes to request all needed resources before starting up. However, this advance knowledge is impossible in many cases. Another way is to require processes to release all their resources before requesting all the resources they will need, but this is also often impractical. The "no preemption" condition may also be impossible to remove as processes must access a resource for a certain amount of time or the processing outcome may be inconsistent.

Finally, the circular wait condition is the easiest to remove. A process may be allowed to possess only one resource at a time, or a ranking may be imposed, eliminating waiting cycles. A hierarchy typically determines a partial order between resources.

6.2.8. EXECUTABLE CONTEXT EXAMPLES FOR SOLVING DEADLOCK AND LIVELOCK

The EC method provides an answer to the modeling and simulation cases of both deadlock and livelock detection between processes. Furthermore, EC enables a systems process aligned with an operational process to execute in chronological order as a sequence of events. These events are in predetermined states that change as the simulation progresses. For example, the state of a phone operator in a process could go from "idle" to "busy" as a call is answered and back to "idle" after the call has been routed.

The EC approach is advantageous when a process is finite because each step can be modeled accurately and the variables controlled to avoid errors. Also, the modeler can control the rate at which the simulation runs - slower to observe individual outcomes or faster to run the simulation multiple times for a distribution of the results.

Using EC to model the system under test would allow the modeler to analyze concurrent systems and operational processes and to stop the simulation where a deadlock/livelock occurs. An algorithm can determine which process relinquishes and which retains control of the resource(s). By modeling the processes and their interactions, the frequency of deadlocks/livelock can be recorded over many simulations and determined if the frequency is acceptable.

In the EC solution for deadlock/livelock detection and resolution strategy, resource requests are granted without considering the potential for deadlock. At appropriately chosen times, a deadlock detection procedure is invoked. If the procedure identifies a deadlock, the deadlock is resolved.

Many algorithms that detect or prevent deadlocks reorder the waiting entities into a non-decreasing list by request size. The algorithms then attempt to find at least one execution sequence that does not result in deadlock. In the case of simulation systems, reordering the waiting entities may violate the reallocation rule of the resource. The reallocation rule decides whether the resource queue allows late arrivals in the queue to pass stalled entities. In the case of multiple-unit requests, this policy may alter the outcome of the deadlock detection by erroneously designating a deadlocked simulation to be free from deadlock.

In the case of manufacturing system real-time control, resolution is achieved by removing a deadlocked entity from the resource (machine) it holds, placing it in a temporary buffer and reallocating the released resource to a waiting entity. The resolution procedure in computer applications typically chooses a set of entities to be aborted and restarted or partially rolled-back to break the deadlock.

Due to the varied nature of deadlocks in a general simulation system, removal or roll-back of entities is not directly applicable while modeling manufacturing systems. Preferably, EC could allow the simulation to automatically recover from deadlock situations without additional burden on the simulation modeler. Also, the ability to run the simulation many times in a short time span allows the modeler to test under different conditions and gauge the effectiveness of multiple solutions.

Another advantage to using EC is the use of a master scenario event list (MSEL). The MSEL is a list of events with corresponding times at which to inject the events into the simulation. Using the MSEL, a deadlock/livelock can be injected at any given point in the simulation to test the system under different situations. This also ensures that a deadlock/livelock does in fact occur rather than waiting for it to randomly occur while testing for solutions.

EC may accurately model the problem of deadlock/ livelock by categorizing then prioritizing the discovered deadlock using information from the initial procedure. This places no additional computational burden for the information required by the prioritization procedure.

In the case of group processing deadlocks, it is possible to develop appropriate prioritization to resolve the deadlocks once the reduction procedure is in place. This is accomplished by automatically displacing some of the entities in deadlock. The procedure is applicable to both categories of permanent deadlocks. It is assumed resolving transient deadlocks are not a logical process in the case of group processing. This is due to the time penalty involved in displacing entities and the duration of the transient deadlocks at the time detection is not known.

6.2.9. DEADLOCK AND LIVELOCK PROBLEM FORMATION

In early problem formulation, EC can test the model design and check for deadlock or livelock, thus prioritizing processes to avoid this situation during actual process execution. If it is impossible to identify deadlock using informal methods, studying aspects of the EC specification can be used. EC uses the DEVS structure as defined below:

M = {X, S, Y, δint, δext, λ, ta} where:

X = set of inputs values

S = set of states

Y = set of output values

δint = S → S internal transition function

δext = Q x X → S external transition function

λ = S → Y output function

ta = S → R time advance function

Equation 1: DEVS equation structure is used in executable context

Simulating a model within the EC method will produce trajectories like the ones shown in Figure 17 below. The figure shows the following information displayed over execution of a federation of models using the EC method:

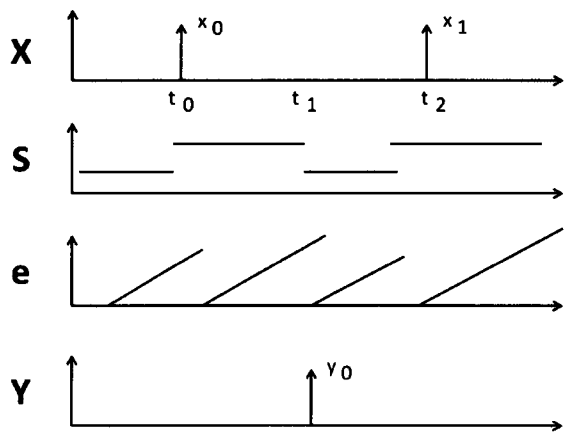


Figure 17: Executable context deadlock and livelock analysis

Deadlock detection and resolution can be managed using mostly the operational and systems state defined within the EC specification. The undesired deadlock condition is simply a system state and operational process. That state, as well as state sequences leading up to it, should be recognized and avoided. First, the culprit states need to be

revealed via simulation. Next, the processes within its states need to be used to reconfigure the models to avoid deadlock or livelock. Initial simulation can uncover any of these situations that arise. During initial simulation, state transitions can be exported as diagrams, matrices, or other machine-readable formats.

Patterns must be discovered that represent deadlock and livelock, as well as the patterns leading up to them. Capturing system states that lead to or currently represent these states must be identified using simulation by transitioning current and other flagged states that lead to these patterns. System states that currently mirror a flagged state transition map will require intervention to break the upcoming situation.

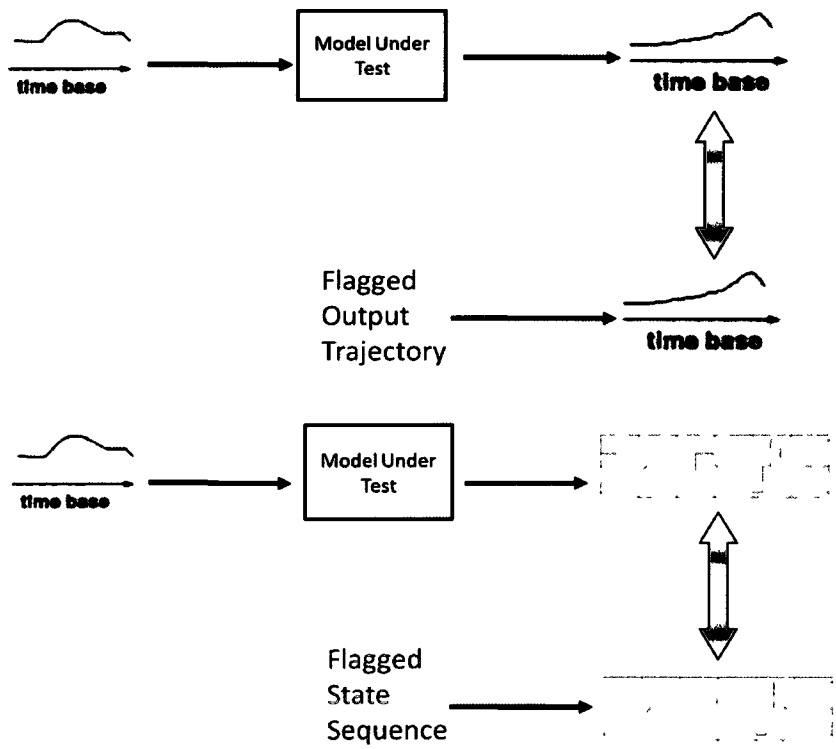


Figure 18: Flagged state transition map

State transition diagrams at atomic and coordination detail levels can be traversed and scanned for static and cyclic patterns that represent or lead to deadlock and livelock situations. Many of the patterns identified can be used to automatically

detect and resolve these issues in later simulations. Specific processes prone to deadlock can use the current state or sequences of previous states to detect, resolve and transition to prevent these situations; examples of this will be described in more detail in section 6.3.

Using EC, algorithms and logic that detect and resolve the deadlock situation can supplement complex models a number of ways. Process priority logic algorithms can be embedded inside deadlock or livelock processes and models. Further, logic could be separated from normal model design and stored in mediation nodes serving as passive monitors. During preliminary phases, flagged state maps and "doomed" patterns can represent atomic models or coordinators (models with children). If a deadlock or livelock is a particular process' state transition, the local flagged state map could be used by that process to prevent the situation. In this case, the process would internally transition differently to prevent the situation.

If the cause of the deadlock is more complex or at a global level, intervention from a higher authority may be needed to control multiple paralleled paths causing the condition. Priority mediation nodes present among the processes can constantly monitor state sequences for their respective responsibility processes and identify those "doomed" for a deadlock. If the models are on a path to deadlock, the priority mediator must intervene by injecting instructional messages to worker models to change their behavior. Additional message interpretation, transitioning logic and input ports must be added, but the mediator design would be less intrusive to original design and would cut down on programming for each participant models or processes. Priority mediation models can plug in and prioritize deadlock and livelock processes.

After preliminary phases discover the condition, a reconfiguration phase will assign various priorities to execution threads. This case may involve situations where a resource recognizes that two threads are competing for it and allow a higher priority thread to execute first. Priority levels of incoming requests can be stored in incoming messages. External transition functions can interpret the message content, transition and output messages accordingly. Should low priority threads be continually shut out,

priority mediators will intervene. Mediators will send messages to standard processes' input ports to manipulate their functionality. Priority mediators could also dynamically set priorities of various threads via "resolve" messages to temporarily allow a low priority thread a chance to execute. Priority mediators must be aware of local and global systems states to have enough intelligence and situational awareness to make these decisions.

7. OPERATIONAL EXAMPLE

In this section, the mechanics of EC and the four step process are described using an operational example. Beginning with the EC simulation framework in section 7.1.1 and its architectural mapping, EC's process is described in detail in sections 7.2.1 through 7.2.4. The results of the operational experiment and how EC proves effective is contained in section 7.3.

7.1. DESIGN OF EXPERIMENT (DOE) FOR OPERATIONAL EXAMPLE

DoE is a systematic approach to investigation of a system or process (Weiss 2009). A series of structured tests are designed in which planned changes are made to the input variables of a process or system. The effects of these changes on a pre-defined output are then assessed (Taguchi 1986, Tamelu 1988).

For EC, the DoE is a formal way to maximize information gained for knowledge generation and to aid in mapping the NCOBP problem formulation and the architecture framework to the MMF. The DoE offers more than "one change at a time" experimental methods as it allows generation of the needed measures that answer the specific questions defined in the problem formulation.

"One change at a time" experiments always carry the risk that the experimenter may find one input variable to have a significant effect on the response (output) while failing to discover that changing another variable may alter the effect of the first (i.e. dependency or interaction) (Taguchi 1988). This often occurs because the experimenter is tempted to stop the test when the first significant effect is found. To reveal an interaction or dependency, "one change at a time" testing relies on the experimenter to carry the tests in the appropriate direction. However, a DoE plans for all possible dependencies (thought experiments) first and then prescribes the requirements to exactly measure these dependencies (i.e. whether input variables change the response on their own, when combined or not at all). In terms of resource, the exact length and size of the experiment are set by the design before testing/experimenting begins.

Managing large system-of-systems with complex integration and interoperability issues is challenging. Functionality and information are not regularly reused, resulting in duplication. In a growing environment consisting of hundreds of interconnected systems, co-existence is difficult to maintain. Despite the success of many individual projects with their local goals, the military continues to face difficulty of incorporating these minor solutions into an enterprise-level portfolio. Moreover, systems and functionality must be syntactically and semantically incorporated into the shared environment.

To address the problems this research has identified, consider the following questions:

1. What is an appropriate approach to make a system specification that is available in the form of a system architecture executable?
2. Can the resulting artifact be federated into an executable context that represents all external systems and can be initialized with the elements describing an operational scenario?
3. Can the relevant measures of performance on the system level and measures of effectiveness on the scenario level be derived from operational requirements?

The EC method was developed for these types of challenges. The Department of Defense Architecture Framework (DoDAF) has served as the common overarching framework for understanding, comparing and integrating architectures across organizational boundaries. As indicated by the literature research, DoDAF architectures are not effective for enabling quick and efficient information flow among complex system-of-systems and decentralized organizations. An architecture describes an organization's missions, structure, business processes, information exchange requirements, system-level infrastructure and other characteristics. Expressing architectures using DoDAF lays the foundation for achieving interoperability. DoDAF enables the alignment of architectures that supports a federated approach. The use of architectures also promotes the sharing, reuse and composability of architecture

components and viewpoints. Various issues involving complex integration and interoperability with large system-of-systems can be managed and resolved using executable architectures.

Once created, architectures are typically used as static descriptions of systems and organizations that depict operational, system and technical viewpoints. In order to fully execute the architectures, simulators must use the dynamic and behavioral aspects described in the DoDAF viewpoints and in the underlying meta-model. Executable architectures induce the dynamic behaviors and provide performance measures for evaluators. DoDAF can be used to describe the functionality of various systems and the missions and test events they participate in. Since EC can exercise the models that represent systems and events, EC not only optimizes these systems and events, but EC also reveals the integration and interoperability problems associated with these systems and events prior to encountering them during live testing. By utilizing EC before testing begins, the military can significantly avoid problems that arise during live testing.

Of course, proper evaluation is a critical aspect of testing and requires additional research. Simulation can help ensure that events or missions represented by architectures are generating the necessary metrics for evaluation. From a planning standpoint, evaluators can define architectural objects of interest, measures, objectives and criteria. Later, those metrics can be extracted via iterative execution using varying conditions and architecture configurations in order to ultimately converge to an optimal solution to best satisfy the objectives.

In short, issues remain with effectively describing what happened during a test when it relates to the execution of an architecture model. Pre-test simulation can also be initiated to reveal what can happen under varying scenarios and conditions. Traceability between architectural descriptions and gathered data could help indicate what happened during the evaluation and analyze why problems were encountered. From an architectural perspective, progress indicators could track evaluation progress and ensure that the evaluation is on the right track. Fatal discrepancies can also be caught before continuing a potentially corrupted evaluation. Information flows among

various producers and consumers according to architectures would also be visually confirmed.

7.1.1. PUTTING EC TO PRACTICE: DEVELOPMENT OF THE EXECUTABLE CONTEXT SIMULATION FRAMEWORK (ECSF)

This research conducts experiments to test the utility of EC. One experiment provided a qualitative example of the entire EC method to attain qualitative measures of Net Enabled Weapon defense systems architecture within its intended operational environment before the system is built. A simulation framework called Executable Context Simulation Framework (ECSF) is depicted below in Figure 19.

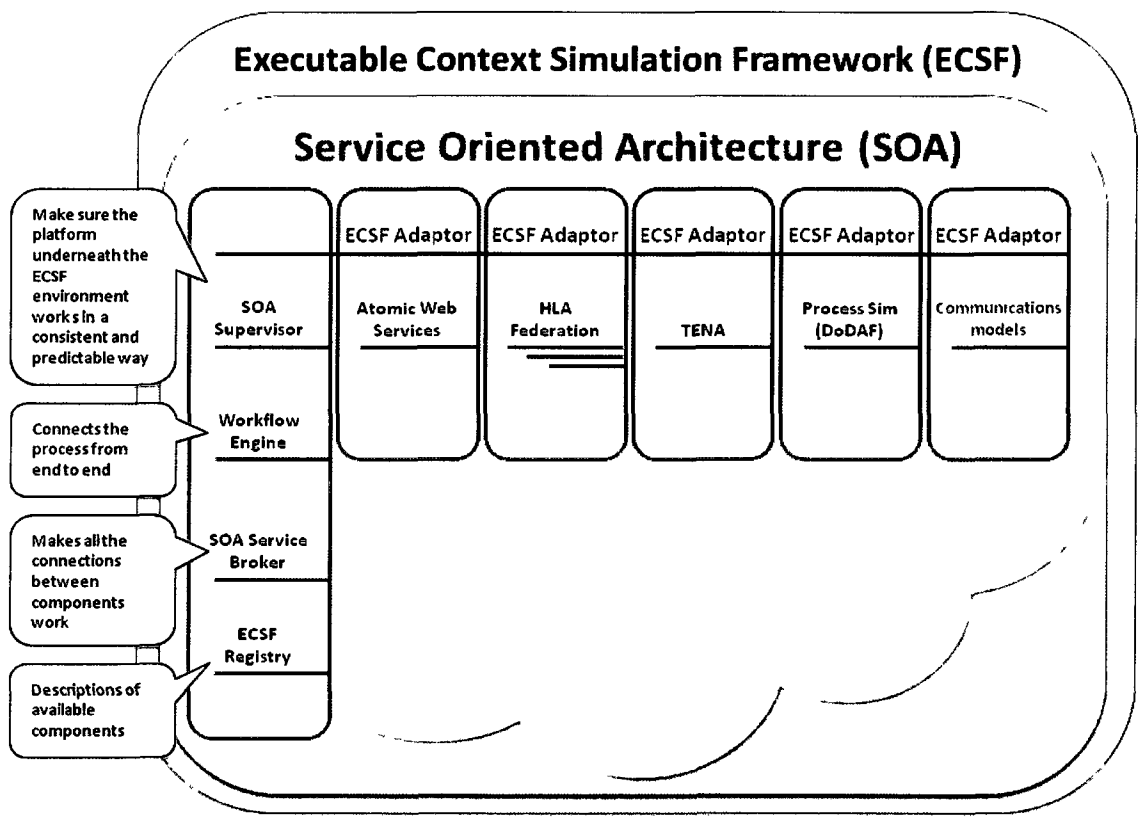


Figure 19: ECSF

The central point of the ECSF resides in executing the simulator as a web service. The development of this kind of framework will help to solve large-scale problems and guarantee interoperability among different networked systems, specifically discrete event system specification (DEVS)-evaluated models. DEVS is one of the most suitable formalisms for the representation of real world systems. Simulating a model involves implementing a behavioral model and running it in the simulator. A simulator is defined as a piece of program that executes the model. Web-based simulation requires the convergence of simulation methodology and web service technology. The capability to run the simulation service is provided by the server side design of DEVS Simulation protocol supported by the latest DEVSJAVA Version 3.1.

Reuse and composability principles will be followed in a number of ways. ECSF open-simulation framework enables connectivity with other simulators during execution. The fundamentals of the ECSF are based on its role in distributed simulations. These fundamentals allow ECSF to send and receive events to and from remote systems during execution. The executable architecture can be reactive or act as a stimulator. ECSF High Level Architecture (HLA) and XML interfaces can create a federation of simulations to further enhance the fidelity of outputs and measures; Figure 20 shows the XML interface for the simulation framework.

Figure 20 also illustrates the process of ingesting information from remote sources, which involved subscribing to topics of information in a publish/subscribe architecture. Upon the receipt of information, the parsing process involved XML parsing or data binding using a technology such as Java Architecture for XML Binding (JAXB). Figure 20 also shows how to establish the web service for the ECSF that incorporates DEVSJAVA as an executable service. This service also converts the static architecture model into the required MMF DoDAF operators.

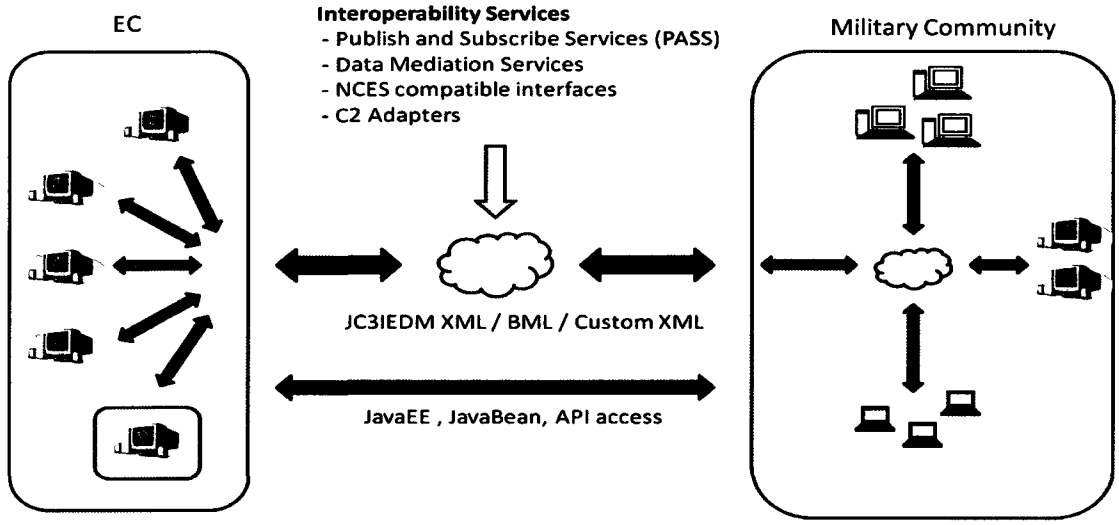


Figure 20: Remote data ingesting

Figure 21 represents the ingestion process; any information relevant to the executing architecture will be injected into the simulation. Outputs or reflections from the simulation could also be sent to remote systems in an event-based manner using the same web service technologies.

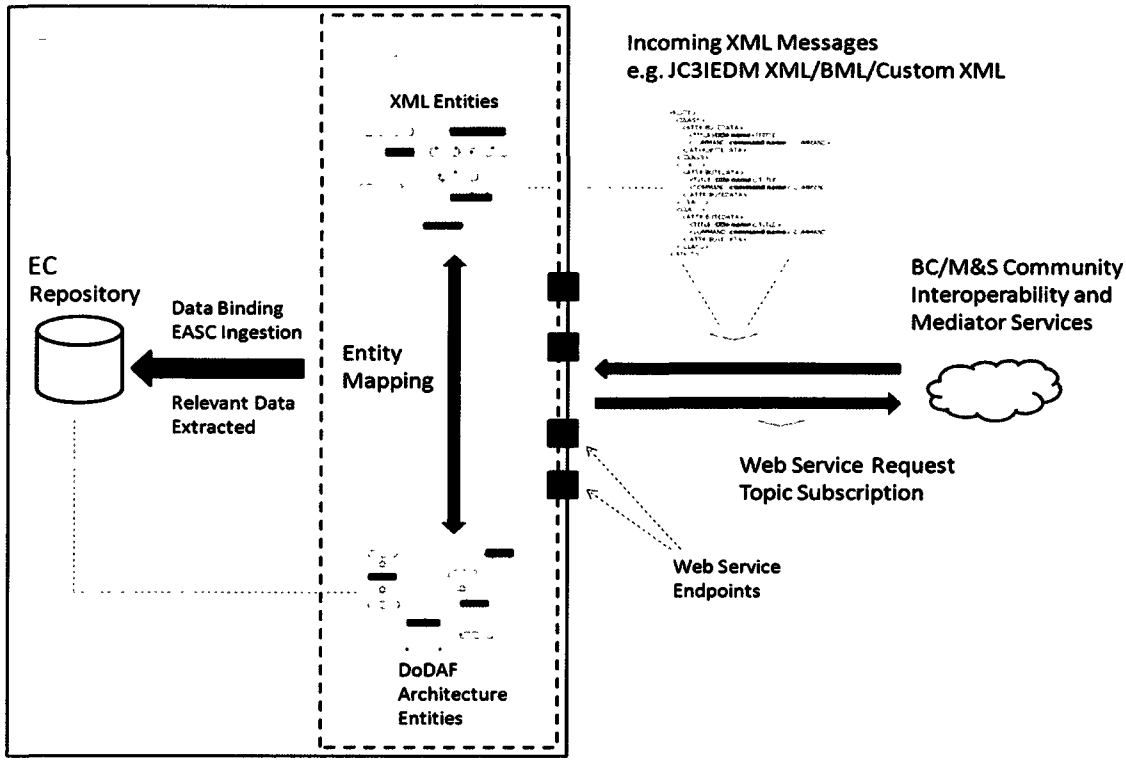


Figure 21: Executable context integration services

DoDAF barriers were considered potential obstacles to achieving objectives. Figure 22 below shows how EC communicates with linked models during an execution and shows parallel execution of all linked simulators over time and messages passed between simulator acts as event triggers. Usually the executable architecture layer triggers activities in the other simulators.

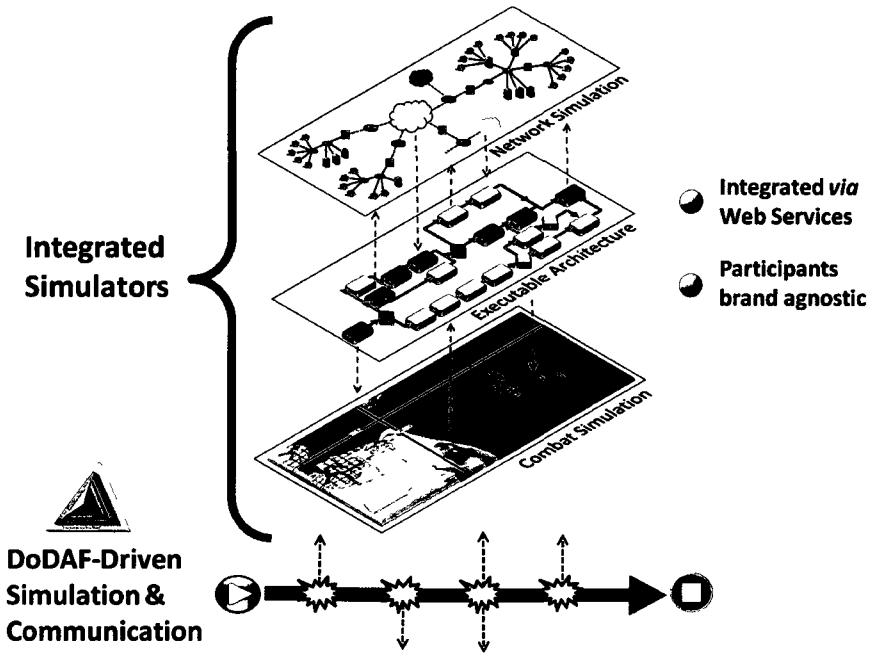


Figure 22: Executable context integration services

Figure 23 shows the EC simulation framework integrated models during the experiment using web services and HLA. Similar to Figure 22, Figure 23 also depicts how the executable architecture triggers activities in other linked models via messages. The executable architecture also receives callback events from other simulators during the course of the simulation.

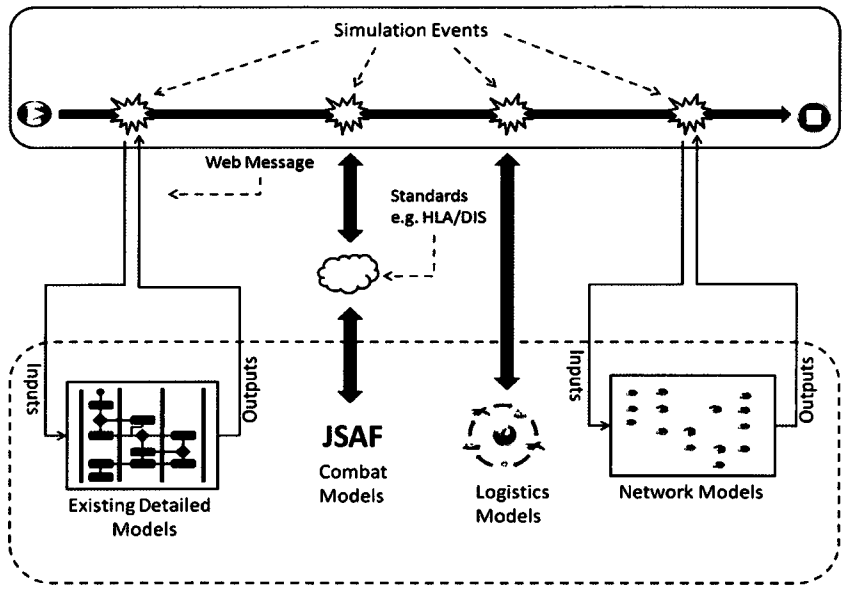


Figure 23: Executable context simulation framework integration with other models

ECSF can use these technologies before and after an execution as well for pre-simulation configuration and post-simulation data extraction and analysis. The fundamentals of the current system are based on being a part of distributed simulation and system, receiving input events from remote systems, and acting as a messaging system among other interoperating systems.

Figure 24 represents the process that was used to map high-level or lower-level statistics generated from models or sub-models to higher level operational goals and form a bigger picture of how the capability is working from a larger scope and how it affects or contributes to the goals of other systems. Figure 24 is a concept diagram of how ECSF can be used to identify problems in the architectures. The resulting artifacts are federated into an executable context that represents all external systems and can be initialized with the elements describing an operational scenario, allowing relevant measures of performance on the system level and measures of effectiveness on the scenario level to be derived from operational requirements while using standard simulation architectures environments and common frameworks.

During the evaluation, the architecture would help drive the simulation by stimulating other systems and reacting to callbacks from artifacts and environment using logic designed in the federation. Afterwards, results and metrics are gathered to be compared to the requirements and goals of the evaluation.

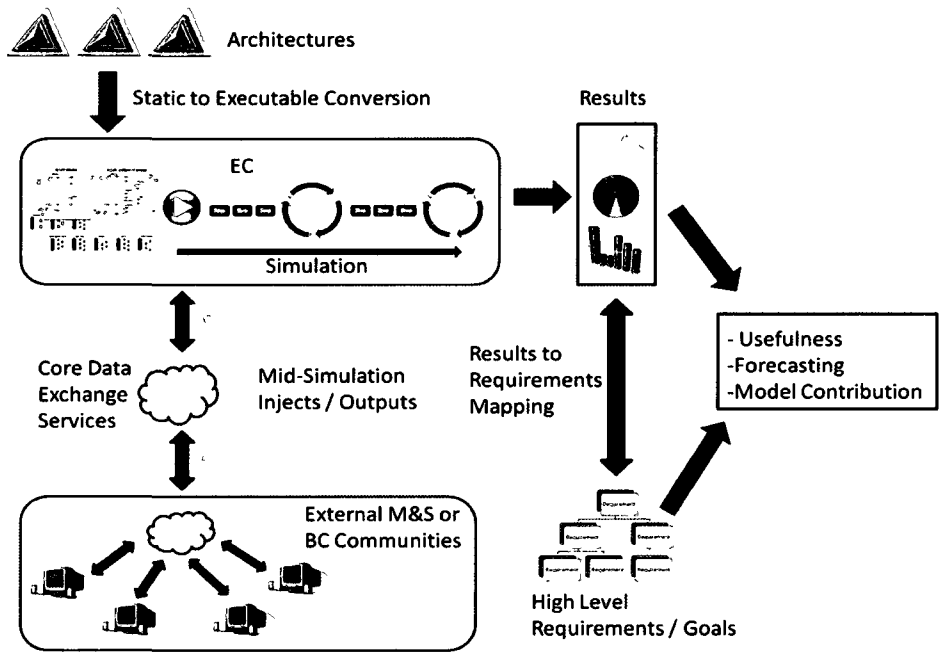


Figure 24: ECSF mapping of models and results

Figure 25 below describes how each model provides its respective results and how ECSF creates traceability between results. This figure shows linked simulators and their respective outputs. Using each simulator alone does not tell the whole story. However ECSF understands the causal relationships between all results across all simulators. For example, users can look at outputs from Operational Activities from the executable architecture components and drill for more data about network transmissions from the network simulator component that occurred during that Operational Activity. ECSF consolidates results.

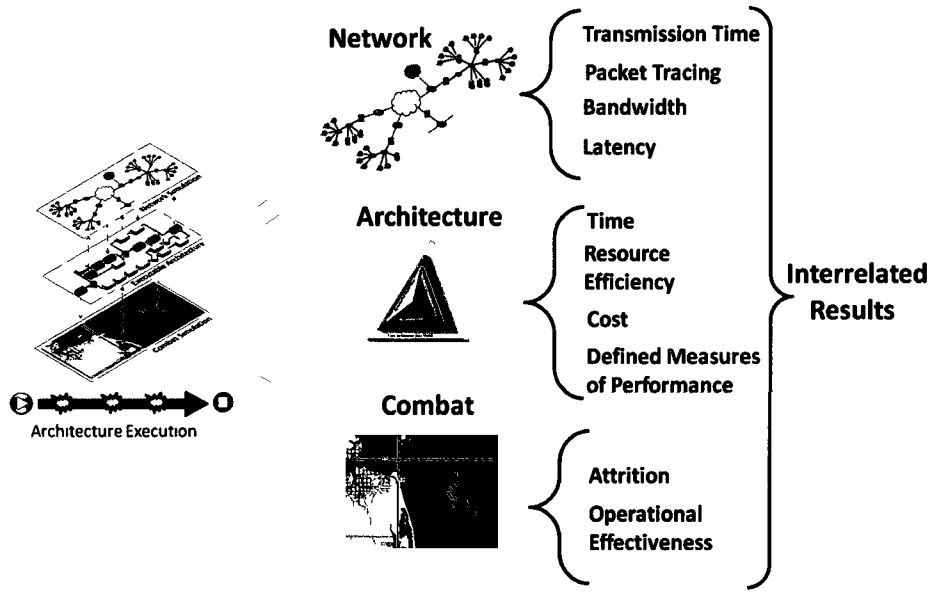


Figure 25: ECSF Interrelated results

Since the major focus of the EC method is used for evaluation, an essential attribute is the usefulness of the results. In anticipation of many unique and specific uses of the EC method, EC was designed for user-defined data collection for any part of architecture and even information architecture modeled outside the architecture such as network modeling for system-to-system interactions. The current EC configuration also allows the customization of individual architecture elements such as operational activities as described in the OV-5.

The OV-5 describes the operations that are normally conducted in the course of achieving a mission or a business capability. It describes capabilities, operational activities, input and output flows between activities, and I/O flows to/from activities that are outside the scope of the architecture.

The purpose of the OV-5 is to: (1) clearly delineate lines of responsibility for activities, (2) uncover unnecessary operational activity redundancy, (3) make decisions about streamlining, combining, or omitting activities, (4) define or flag issues, opportunities, or operational activities and their interactions that need to be scrutinized

further, (5) provide a foundation for depicting sequencing and timing, and (6) identify critical mission threads and operational information exchanges by annotating which activities are critical [DoDAF v1.5, Volume II: Product Descriptions, 23 April 2007]. The ability to insert logic capable of modeling the realistic activities of an OV5 node could also provide an assessment of the system's usefulness from a modeling perspective.

Another factor to review the method is the ability to simulate different types of scenarios in many different ways. The generic simulation component of EC is based on standardized fundamental M&S principles, which allows it to simulate many types of models other than typical flow models. Its flexibility does not confine its use to one specific purpose, allowing it to execute architectures in a variety of ways unique to the user's preferences. Some commercial M&S tools can only simulate typical flow models for specific purposes like queue studying. EC can be measured by its ability to accurately execute tightly- or loosely-coupled systems while producing useful results. EC could also be evaluated on its ability to integrate and simulate architectures, processes and models. Figure 26 shows how two architectures are integrated using EC generic DEVS modeling approach. The figure shows how architectures are difficult to integrate in their pictorial format even if DoDAF is the common denominator, which is supposed to allow integration. However, once represented in DEVS format, architectures can more easily integrate via DEVS coupling.

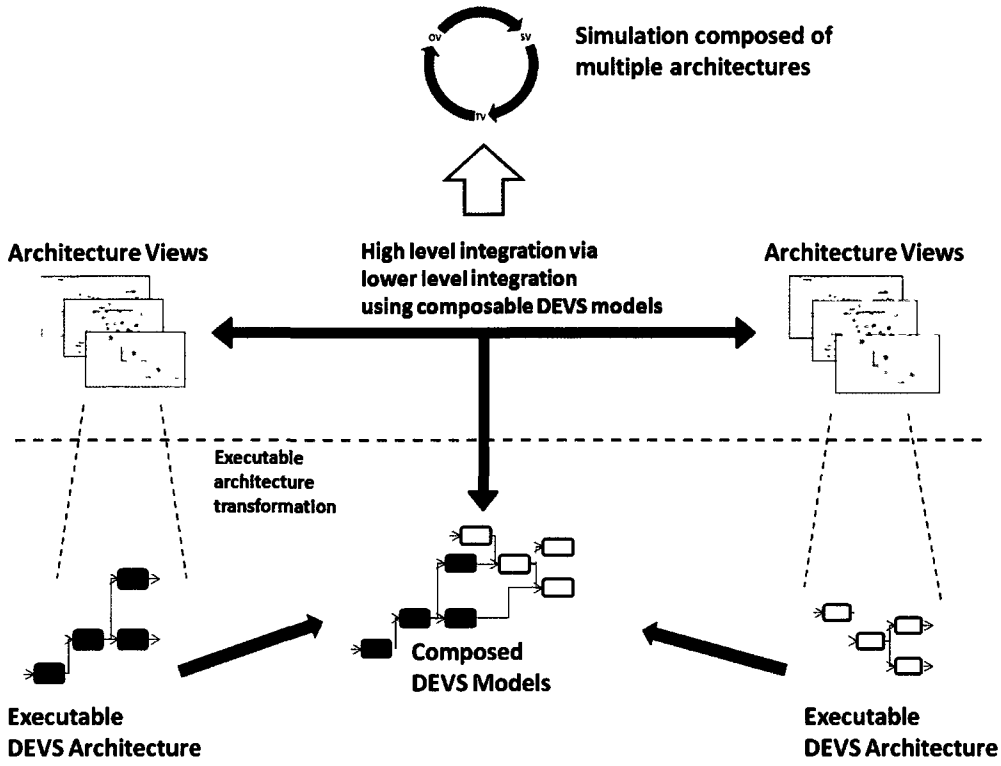


Figure 26: Executable context architecture integration

The plan is to use a variety of techniques when attempting to evaluate the architectures. An initial preference would be to use output evaluation and compare architecture execution results test data that may be available, but other evaluation techniques will also be considered and utilized in order to evaluate the architectures. This type of output testing is likely to use statistical techniques to compare output data and trajectories from the model with output data and trajectories from the system and study correlation. Although black box testing using input/outputs mappings could be used for basic evaluation of models, the ECSF doesn't have to narrow its analysis to the high level input/output behaviors of an architecture. The ECSF models every part of an architecture and would allow the decomposition of an architecture into lesser modeling parts with their own behaviors and analysis, some which may have their own individual data or detailed subject matter expertise. This technique follows a bottom-up approach where lower-level sub-models are cumulatively evaluated up to the highest level.

Integration would be evaluated for models at the same level. Testing would be repeated until all system components, sub-models and the entire model have been integrated and tested. Evaluation could be assisted by confirming individual parts of architectures from the ground up.

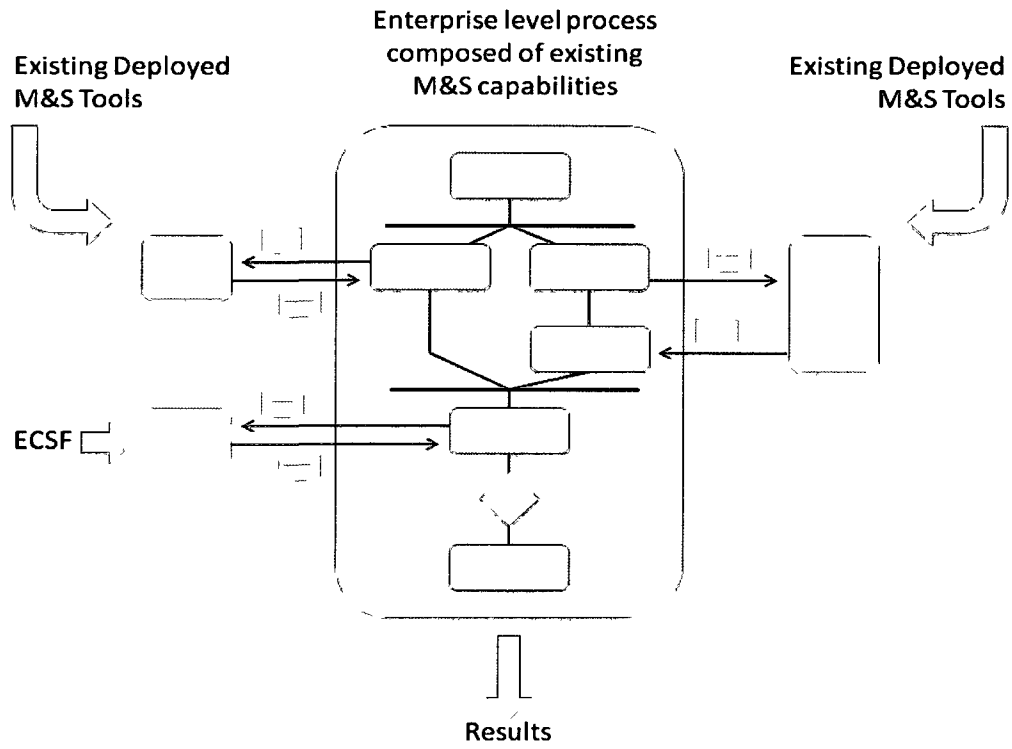


Figure 27: ECSF as part of an enterprise-level process

The Joint Semi-Automated Forces (JSAF) 2007 in Figure 28 provided the operational context and executable architectures to represent the mission and flow of information, and the network models provided the simulated information flow over operational networks for the NEW example.

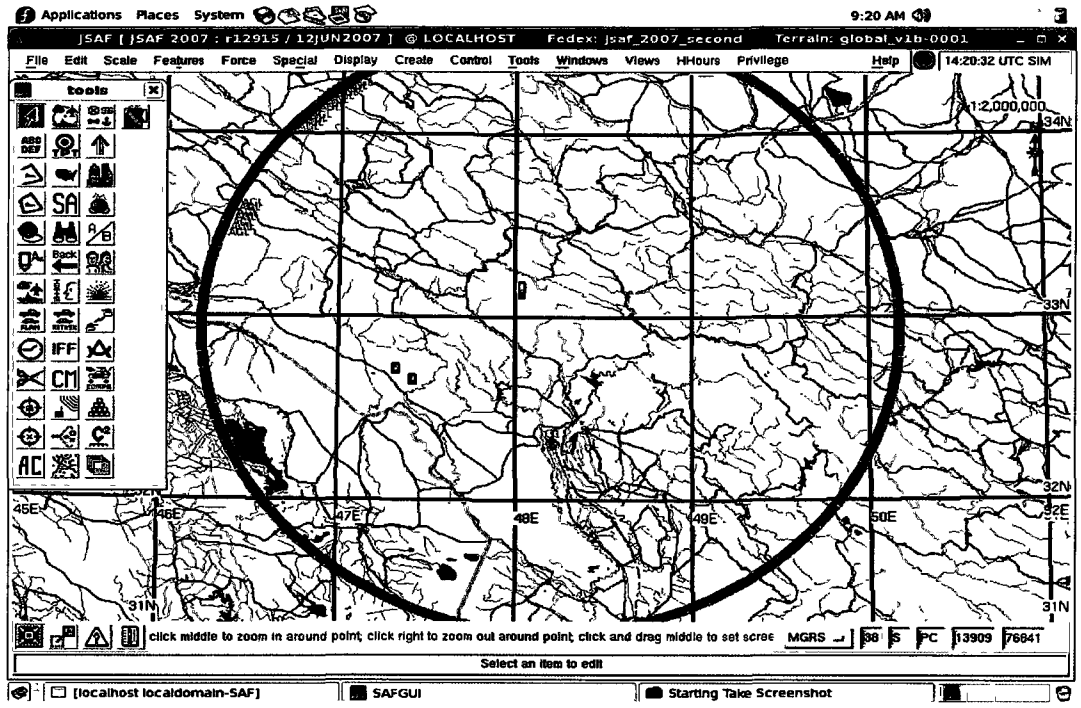


Figure 28: JSAF

The intercommunication provided by the federation allows events in federates to trigger events in other federates, which creates a flow of execution among simulators. Examples include combat simulator events triggering reactive process flows in the executable architecture or information exchange sequences in the architecture triggering network transmissions in the network simulator. Federated simulators are not limited to combat and network models and can re-use any existing simulator of interest.

7.2. EXECUTABLE CONTEXT: FOUR-STEP METHOD IN PRACTICE

EC begins with systems architecture and an operational context for obtaining utility factors (metrics). Figure 29 shows the 4 steps of the EC method. In the figure below, mission requirements (MR) and operational requirements (OR) describe the context; system requirements (SR), functional requirements (FR) and capability requirements (CR) describe the system with both comprising the mission blueprint. External systems (ES) are introduced in step 4.

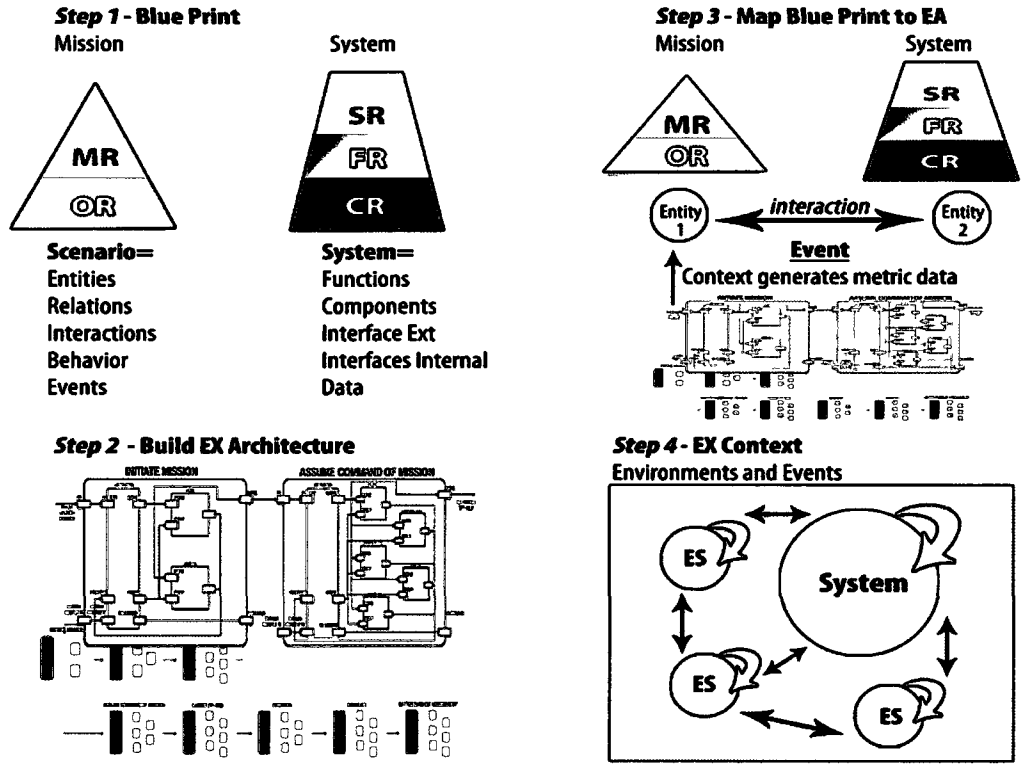


Figure 29: Executable context's four steps for evaluating targeted system-of-systems

7.2.1. STEP 1: DEVELOP THE BLUEPRINT EXAMPLE FOR JOINT CLOSE AIR SUPPORT (JCAS) AS IT RELATES TO a NET ENABLED WEAPON (NEW)

Step 1 for the JCAS experiment is to map the mission and operational requirement with the systems, functional and capability requirements as designed in the architecture. This requires a great amount of data gathering to bind the problem and answer how to make a system specification that is available in the form of a system architecture executable. How can the resulting artifact be federated into an executable context that represents all external systems and can be initialized with the elements describing an operational scenario, and how can the relevant measures of performance on the system level and measures of effectiveness on the scenario level be derived from operational requirements?

During JCAS operations, a target is selected and assigned to JCAS aircraft. JCAS targets may be fixed, relocateable, or mobile and are often time-sensitive. Information is extremely perishable under these conditions. Much of the information aircrews need to attack a target is not available when the mission is assigned or may change while en route to the target. In this environment, real time or near-real time information, through situational awareness, reliable communications, and effective Command and Control is absolutely critical for success.

Targets may present fleeting opportunities where delays could permit the enemy to escape or maneuver to an advantageous position. Immediate JCAS may be in response to situations where the supported unit has encountered a force beyond its capabilities and an immediate response is necessary for success. JCAS systems must permit rapid assessment of the situation and the ability to quickly redirect efforts.

Sensors, which may include Intelligence, Surveillance and Reconnaissance, JCAS aircraft, Joint Terminal Attack Controller (JTAC), or ground force capabilities, track the target location and provide continuous updates to the applicable network. The weapon is then released using the best location information available at the time. During weapon flight, it is possible for the initial target location, as well as the actual target itself, to change. This is relatively common for mobile targets encountered in a JCAS environment. The weapon's impact point is adjusted via In-Flight Target Updates (IFTUs) and the weapon proceeds or is guided to the updated location. Sensors may continuously update the target location throughout the weapon flight.

With network-enabled weapons (NEW), the launcher could engage the target and, if necessary, allow the JTAC to supply IFTUs to the weapon. This is beneficial if the target is no longer discernable from the air. Figure 30 depicts a current JCAS operation. Either non-precision-guided munitions are used or the on-the-ground JTAC guides the weapons using another type of system. Figure 31 depicts the same JCAS operation using NEW. The JTAC will be able to target the weapons using the NEW implementation messages. The sensors onboard the weapons will provide additional guidance.

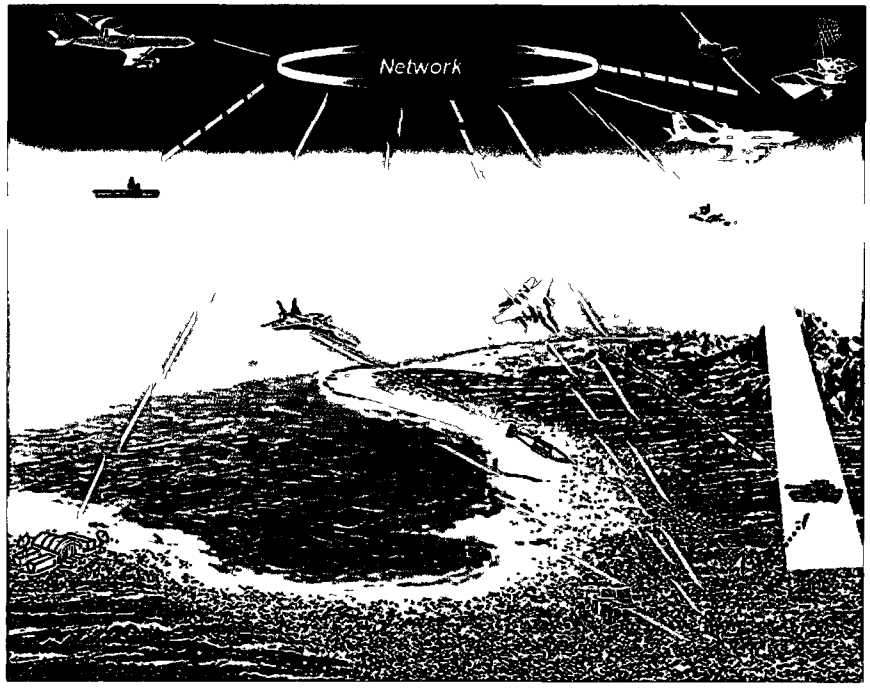


Figure 30: JCAS operation without NEW

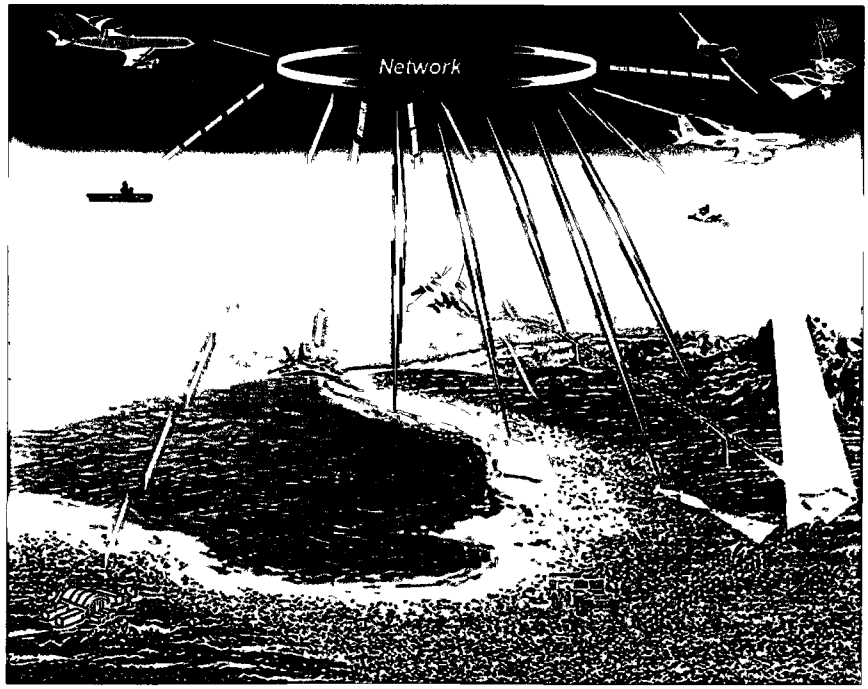


Figure 31: JCAS operation with NEW

The JCAS operation without the Net Enabled Weapon (NEW) systems architecture or the "as is" architecture in Figure 30 represents the baseline model. Figure 31 depicts the same JCAS operation using NEW, or the "to be" architecture. The ability to use the EC method in both cases is beneficial to generate the needed measures to define measures, operational systems architectures and visual tools for capturing systems and operational requirements, decision and operational analysis of external systems which are needed to aid in the development of the SoS to establish the context of the system being evaluated based on the experiment preconditions and assumptions generated during the problem formulation.

7.2.1.1. PRECONDITIONS OF THE JCAS IMPLEMENTATION

This experiment assumes the following preconditions. On orders from Division Headquarters, the Brigade Combat Team (BCT) continues its advance to the objective. Pre-planned Close Air Support (CAS) sorties were allocated by division to the BCT to provide CAS. One of the brigade's battalions has entered a troops-in-contact (TIC) situation and has asked for one of the allocated CAS missions to engage a target: one T-72 tank is in the open. Geographical terrain is flat and obstruction-free. The CAS mission has already checked in with the Air Support Operations Center co-located with the Corps Main Command Post and is currently en route to a holding position 20 NM south of the TIC, at which point it checks in with the Brigade Tactical Air Control Party.

The following assumptions were made regarding the experiment tied to the problem formulation as described in the prior section:

- Resources are 100% available (platforms, sensors, network).
- Flights of fighters are on-station 20 miles away from the target.
- Pilot will not download data to the weapon until clearance to engage is received.
- The weapon functions properly.
- JTAC is on the ground supporting battalion operations.
- JTAC found, fixed, identified and tracked targets.

- Battalion FSO assigned CAS against target and informed the JTAC of the flight.
- Rules of Engagement/Airspace/Ground deconfliction process produces no conflicts.
- Brigade Commander approves the CAS mission.
- Sorties are on the Air Tasking Order in support of the BCT.
- ATO specified time period is 0600 to 0559 ZULU (24 hour period).
- CAS is en route to the BCT Area of Operations upon request for CAS.
- JTAC personnel are trained to control Type 1, 2, and 3 CAS missions and set up their PRC-117F secure radio transmitter, laser range finder, laptop computer, and GPS receiver.
- The airspace is deconflicted/cleared for the CAS mission.
- Digital 9 line includes target coordinates and elevation, attack parameters for bomb impact, and exclusion zones.

Figure 32 below represents the Entity Relationship Diagram (ERD) for the JCAS implementation; it is an abstract and conceptual representation of data. Entity-relationship modeling is a database modeling method used to produce a type of conceptual schema or semantic data model of a system, often a relational database and its requirements in a top-down fashion (Chen 1976).

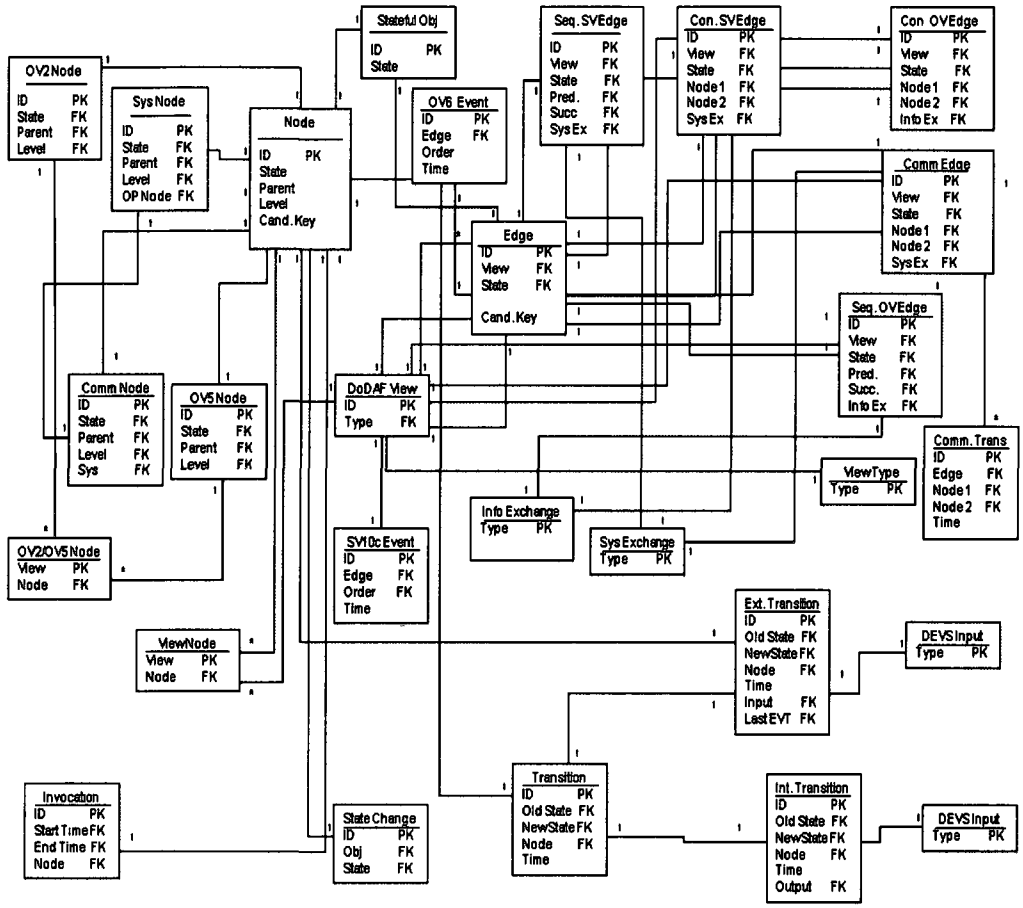


Figure 32: Experiments ERD

Figure 33 below represents the Operational Viewpoint or the OV-6c: Event-Trace Description.

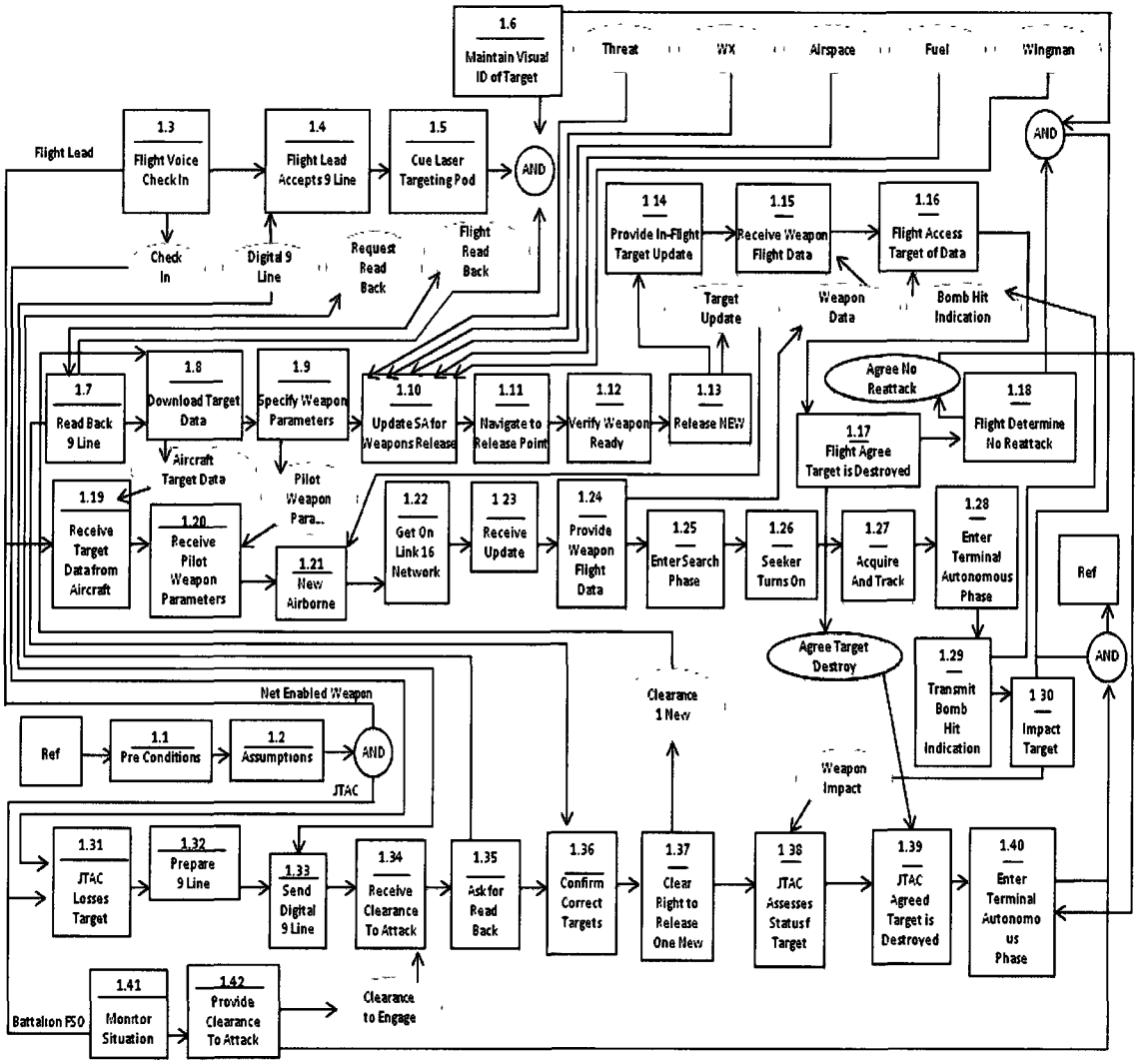


Figure 33: Experiments OV-6C

The OV-6c provides a time-ordered examination of the resource flows as a result of a particular operational context. Each event-trace diagram should have an accompanying description that defines the particular situation. Operational event/trace descriptions (sometimes called sequence diagrams, event scenarios, or timing diagrams) allow the tracing of actions in a scenario or critical sequence of events. The OV-6c can be used by itself or in conjunction with an OV-6b State Transition Description to describe the dynamic behavior of activities. The intended usage of the OV-6c includes:

- Analysis of operational events.

- Behavioral analysis.
- Identification of non-functional user requirements.
- Operational context.

Detailed description: The OV-6c is valuable for moving to the next level of detail from the initial operational concepts. An OV-6c model helps define interactions and operational threads. The OV-6c can also help ensure that each participating operational activity and location has the necessary information it needs at the right time to perform its assigned operational activity.

The OV-6c also enables the tracing of actions in a critical sequence of events. OV-6c can be used by itself or in conjunction with OV-6b State Transition Description to describe the dynamic behavior of business activities or a mission/operational thread. An operational thread is defined as a set of operational activities with sequence and timing attributes and includes the resources needed to accomplish the activities. A particular operational thread may be used to depict a military or business capability. In this manner, a capability is defined in terms of the attributes required to accomplish a given mission objective by modeling the set of activities and their attributes. The sequence of activities forms the basis for defining and understanding the many factors that impact on the overall capability. Table 4 below uses the data from the ERD and OV-6c to map the blueprint developed to the MMF for metrics gathering and the NCOBP problem formulation as shown in Figure 34 below.

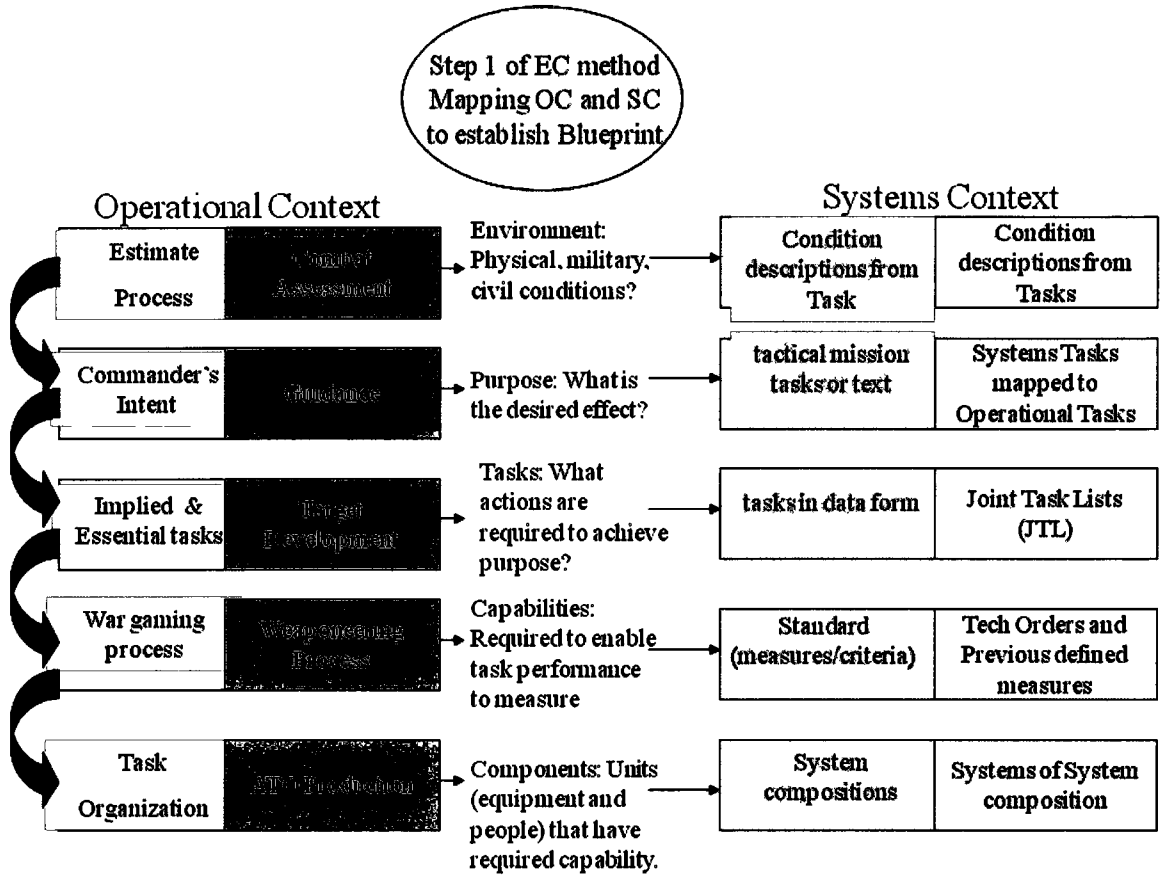


Figure 34: Blueprint mapping

7.2.2. STEP 2: BUILD AN EXECUTABLE ARCHITECTURE

Step 2 is to build an executable architecture that composes the architecture models and simulations into distributed or concurrent systems. This executable architecture was built and simulated using the DEVS/JAVA formalism. As expected, the formalism was able to address discrete event and time-stepped simulation and was generic enough to address all unexpected issues encountered during the process of making an architecture executable or integrating models using web services.

Unlike earlier executable architecture prototypes, the DEVS simulation captures considerably more statistical data. Each element in the architecture, whether a composite process or a particular system node used within a process, is accompanied by an experimental frame which serves as a monitor throughout the simulation. DEVS

simulation monitors and records all time-stamped events of nodes throughout the simulation. From there, recorded information may include resource utilization when a node is invoked. Afterward, the monitors are compiled and the information can be studied. Monitors can be configured to record any type of information of interest to analysts.

As discussed in Section 4.1, DUNIP is leveraged to aid in the development of an atomic model, which is an irreducible component in the DEVS framework that implements the behavior of a component. It executes the state-machine and interacts with other components using its defined in-ports and out-ports. Each such atomic class has its own simulator class. A network of these atomic models constitutes a coupled model that maintains the coupling relationships between the constituent atomic components. The contained services become the DEVS atomic models; research is still ongoing to specify the logic behavior in atomic models. DUNIP provides the ECSF with the process that uses the DEVS formalism as a basis for automated generation of models from various requirement specifications and realization services. DEVS is inherently based on object-oriented methodology and systems theory, categorically separates the model, the simulator and the experimental frame, and has been used for systems modeling and simulation over the years.

The DEVS decoupling implementation also allows real time execution and discrete-event execution where the architecture can be executed much faster than real time. Figure 35 gives an example of decoupling with DEVS and the creation of an atomic model.

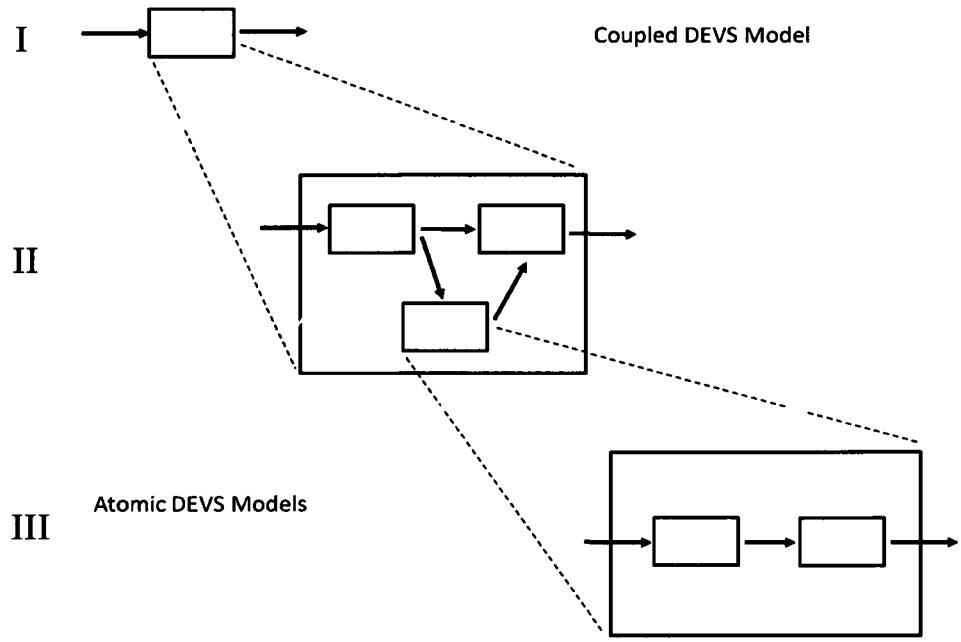


Figure 35: Decoupling example

Figure 35 depicts DEVS models at multiple levels of abstraction. The numbered tiers represent the different levels. The figure shows how models can be extended by sub-models at lower levels using DEVS coupling. One of the significant developments of the ECSF is the masking of a coupled model as an atomic model. What this implies is that we have an abstraction mechanism by which a coupled model can be treated as a black box and can be executed like an atomic model. In other words, a coupled model now has a state machine similar to that of any atomic model. In contrast to the DEVS hierarchical modeling, where a coupled model is merely a container and has corresponding coupled-simulators (Figure 35), now it is considered an atomic model with lowest level atomic simulator (Figure 36).

Using DEVS coupling, ECSF supports a pluggable architecture and can combine multi-resolution DEVS models from multiple sources. Coupling provided by DEVS leads to the coupling of architectures.

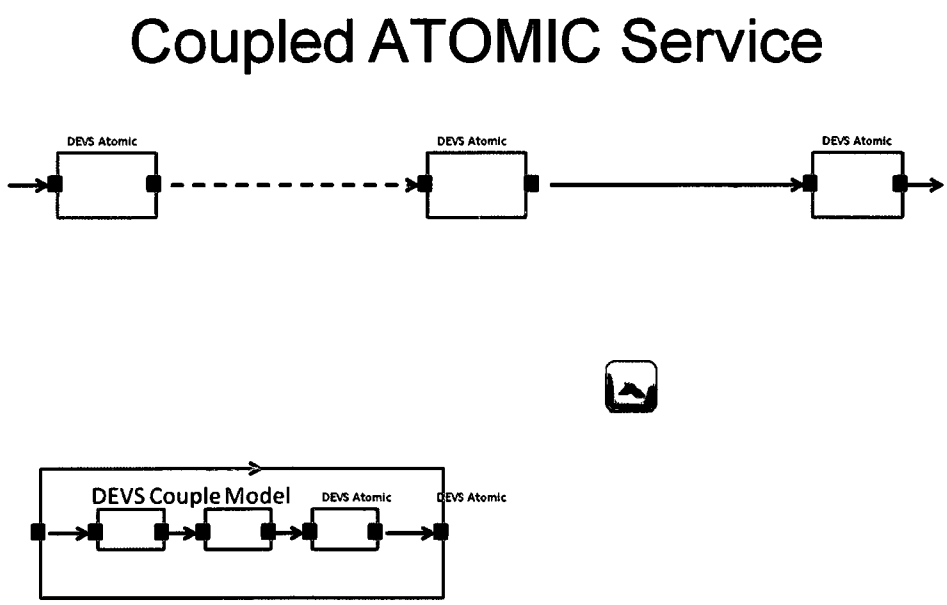


Figure 36: Atomic DEVS models in executable context

Figure 36 shows how an atomic DEVS models could be extending with addition sub-models using external data sources. EC uses DEVS coupling to support a pluggable architecture.

Figure 37 gives a depiction on how ECSF managed the hierarchically structured systems and supports the coupling and composability of atomic models.

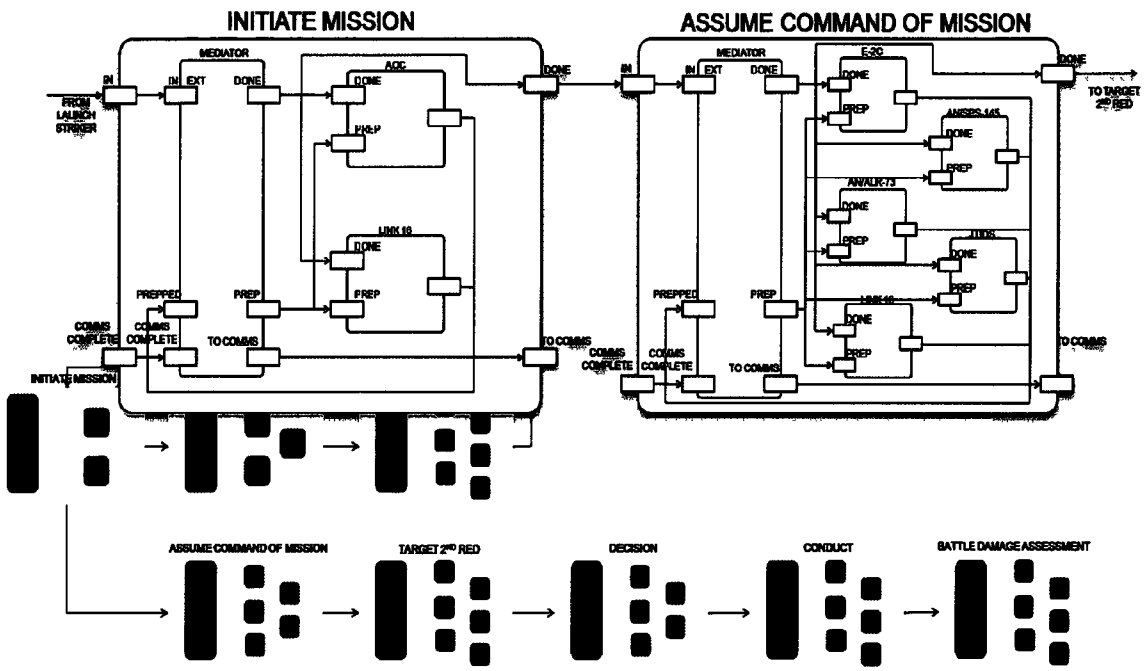


Figure 37: ECSF coupling and composability of atomic models for CAS implementation

It is formed or composed by mapping the components of the operational processes onto the processes of the physical system as shown in Figure 38.

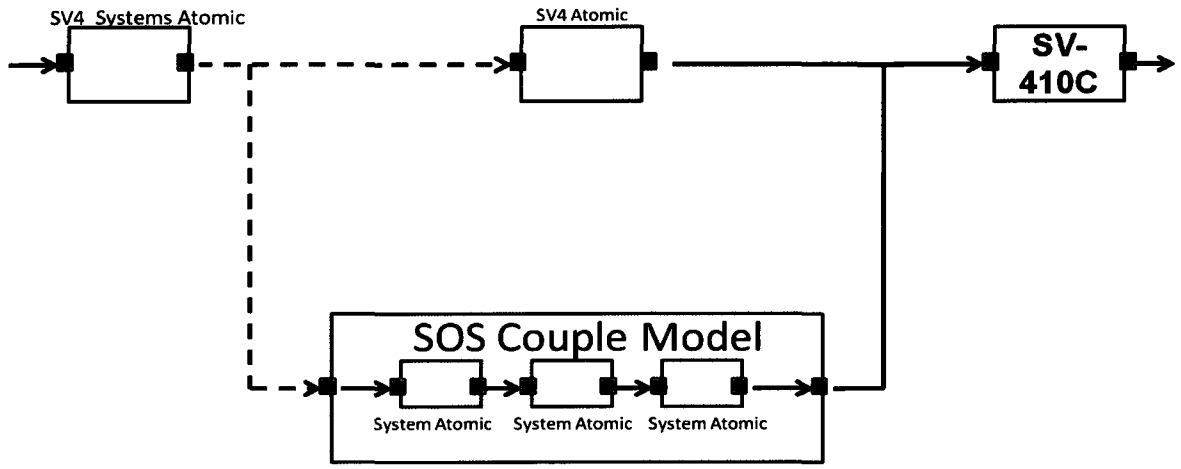


Figure 38: Step 2 of the executable context method – decomposing the systems processes with other systems in the operational process to develop the hybrid view SV-410C

This process enabled the creation of a hybrid systems view SV410c, which combines the Systems/Services Functionality Description (SV-4), documenting system functional hierarchies and system functions and the system data flows between them; the Organizational Relationships Chart (OV-4), which represents command, control, coordination and other relationships among organizations, and the Systems/Services Event-Trace Description (SV-10C), which provides a time-ordered examination of the system data elements exchanged between participating systems (external and internal), system functions or human roles as a result of establishing context as described by the EC method that extracts architecture elements to create the hybrid SV-4/10c for the executable context simulation framework (ECSF).

Each event-trace diagram has an accompanying description that defines the particular situation. SV-10c in the Systems and Services View may reflect system-specific aspects or refinements of critical sequences of events described in the OV-5 Operational Activity Model as illustrated in Figure 39. This includes activities, relationships among activities, and inputs/outputs. In addition, overlays can show cost, performing nodes or other pertinent information of the JCAS example.

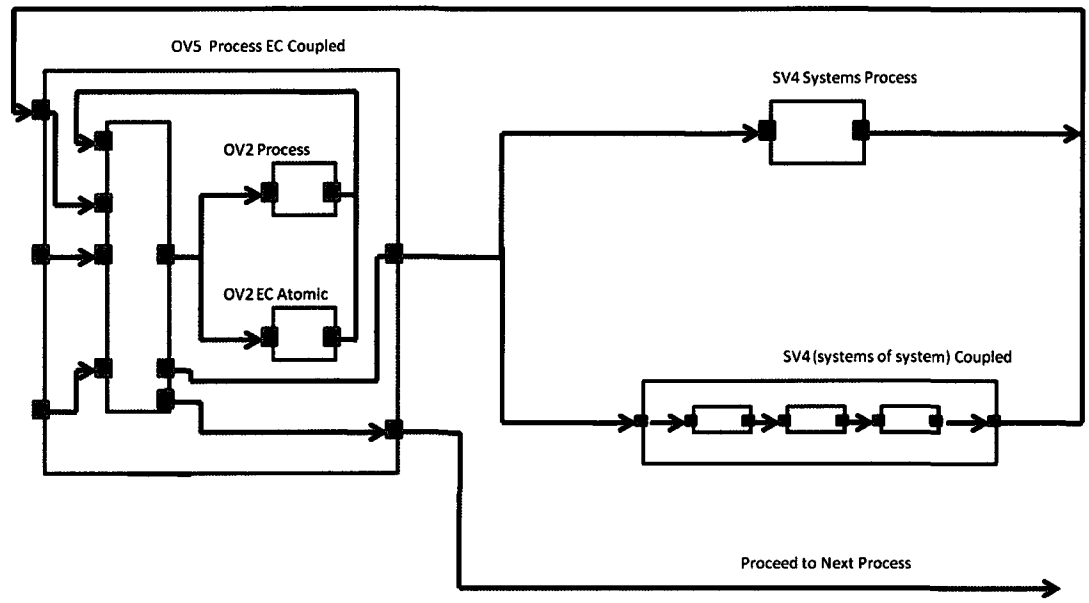


Figure 39: Executable context method step 2b – build executable architecture and compose systems processes with operational processes into the Operational Activity Model OV-5

7.2.3. STEP 3: MAP EXECUTABLE ARCHITECTURE TO THE BLUEPRINT

Step 3 is to map the executable architecture to the blueprint of the JCAS example as shown in Figure 40 below. This allows the use of an M&S environment to capture the measures needed to evaluate the architecture's ability to support its intended purpose.

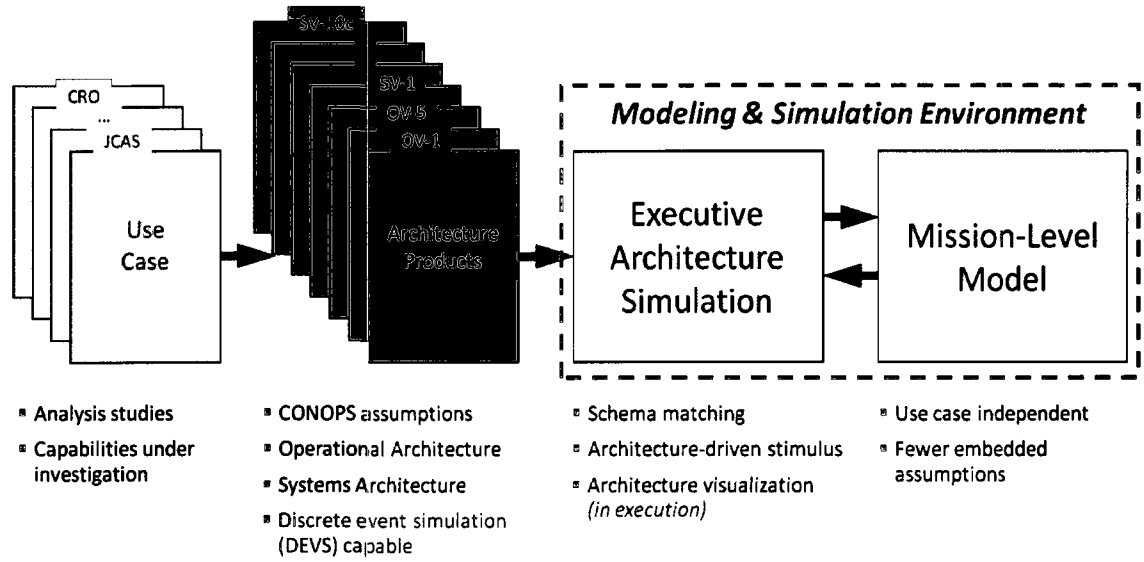


Figure 40: Step 3 – map executable architecture to the blueprint

Ultimately, the structural and behavior characteristics of DoDAF must be represented in DEVS in order to be executed. Architectures are first read in their native format, then placed in an internal data model, and next translated into executable models. First, the architecture must be read and interpreted. This is the uncertain step because it involves third-parties. Both the completeness and format of the architecture are addressed. From a DoDAF standpoint, many of the entities, relationships, and data types of DoDAF data models are able to translate directly to DEVS components. Many of the structural properties of DoDAF translate nicely into an executable DEVS format. For example, a simple graphical representation of a DoDAF operational activity diagram closely resembles a basic chain of coupled atomic DEVS models. Although DoDAF is capable of associating logic, criteria, and timing information with many of its model components, some architecture builders do not include these details and define only the structural properties of the architecture. In cases where executable parameters are not available in the native architecture, parameters will be assigned default values during the initial executable architecture generation. Of course these values could be altered later once the architecture has been ingested. Missing information mostly

includes timing information and criteria for decision-making models. If this information is represented pictorially in architectures, the rules, conditions, and criteria will be reformatted into a machine readable format prior to ingestion.

Recent improvements in the DoDAF data models have made it easier to store this behavioral information in a machine-readable format. Figure 41 below shows a simple operational activity diagram with the data types from the DoDAF Meta-model (DM2) assigned to major components. This is an example of how DoDAF's structural properties translate nicely into DEVS components that can represent the methodology within DoDAF and was implemented in the JCAS example.

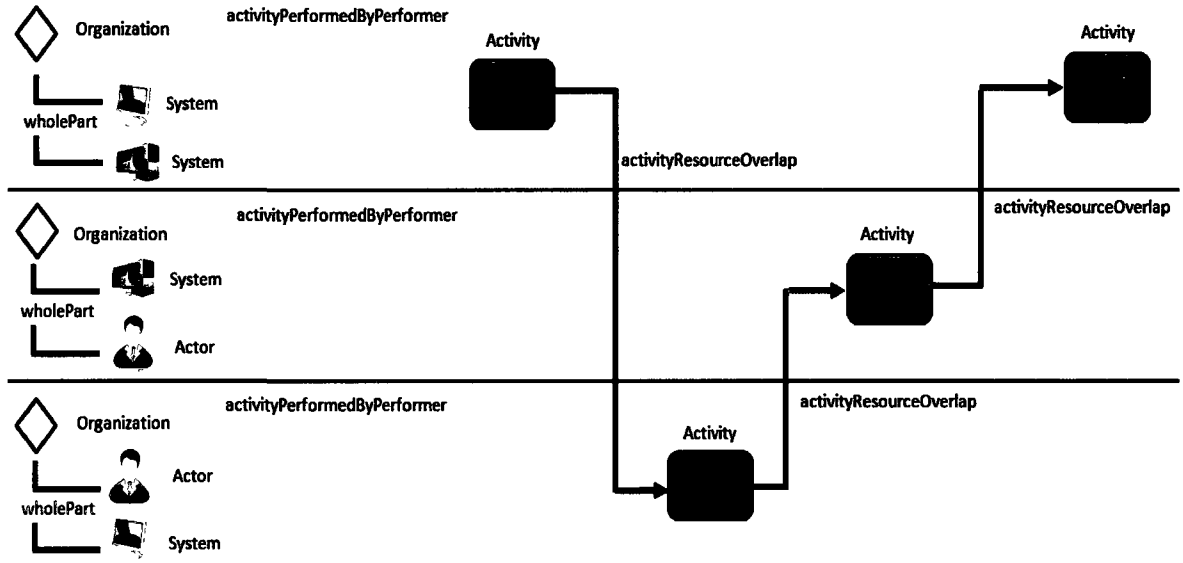


Figure 41: Simple operational activity diagrams

The above model can have countless DEVS representations in DEVS. A simple representation could include a chain of coupled atomic operational activity DEVS models with couplings to their respective resource DEVS models in the swim lanes.

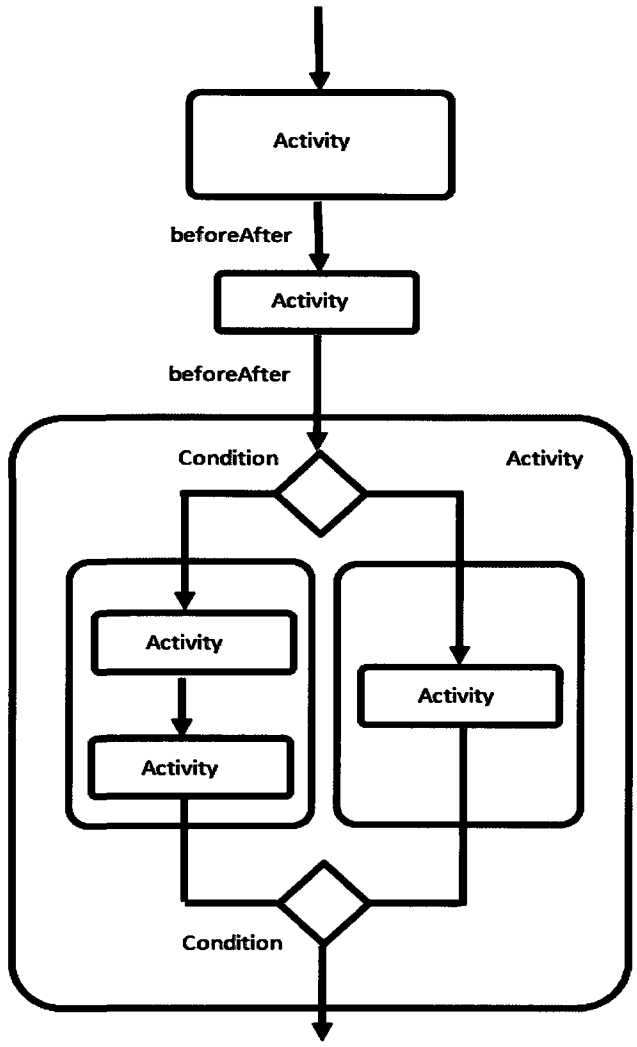


Figure 42: Storage of executable parameters

Figure 42 above demonstrates the improved richness of standard DoDAF data models and their ability to store many required executable parameters, allowing the executable architecture engine to be as parametric as possible. Aside from the architecture completeness, DoDAF architectures are stored in multiple third-party formats. Architectures are typically formatted in pictorial formats, and need to be reformatted into a machine-readable format like Microsoft Excel. Other cases involved DoDAF architecture being represented in XML schemas like Core Architecture Data Model (CADM) and the newer DoDAF Meta-model (DM2).

First, an ingestion component must parse the machine-readable format of the architecture and put it into the internal data model. The code block in appendix A shows how the internal data model is used to generate DEVS models for each of the architecture components. The class is only a template and has been trimmed for simplicity. The class basically loops through and visits every architecture component in the internal data model and generates its DEVS counterpart while generating various ports and coupling everything together using DEVS couplings.

The internal data model is used to generate DEVS models because it simply holds both the structural and behavioral properties needed for the models. Once the DEVS models have been generated, the engine advances using adjustable time-step. The propagation of the engine "ticking" throughout the entire executable architecture is handled by the DEVSJAVA libraries. DEVS model templates contain various collections points to collect information about the architecture during execution. The code block in appendix B shows a template for an Operational Activity DEVS model. The class is only a template and has been trimmed for brevity, but it does contain all the major functions present in all DEVS models in DEVSJAVA.

7.2.4. STEP 4: FEDERATE ALL STEPS INTO AN EXECUTABLE METHOD

Step 4 federates and amalgamates all steps of the method into an executable method as shown in Figure 43 below. Figure 43 outlines how the artifacts and models can be federated into an executable context that represents all external systems and can be initialized with the elements describing an operational scenario and allows relevant measures of performance on the system level and measures of effectiveness on the scenario level to be derived from operational requirements while using standard simulation architectures environments and common frameworks. The ECSF enables the integration of the independent models into an engineering method that allows harmonization of System and operational architecture as an executable. This research also allows for an executable federation that represents all external systems and can be initialized by systems and operationally relevant data.

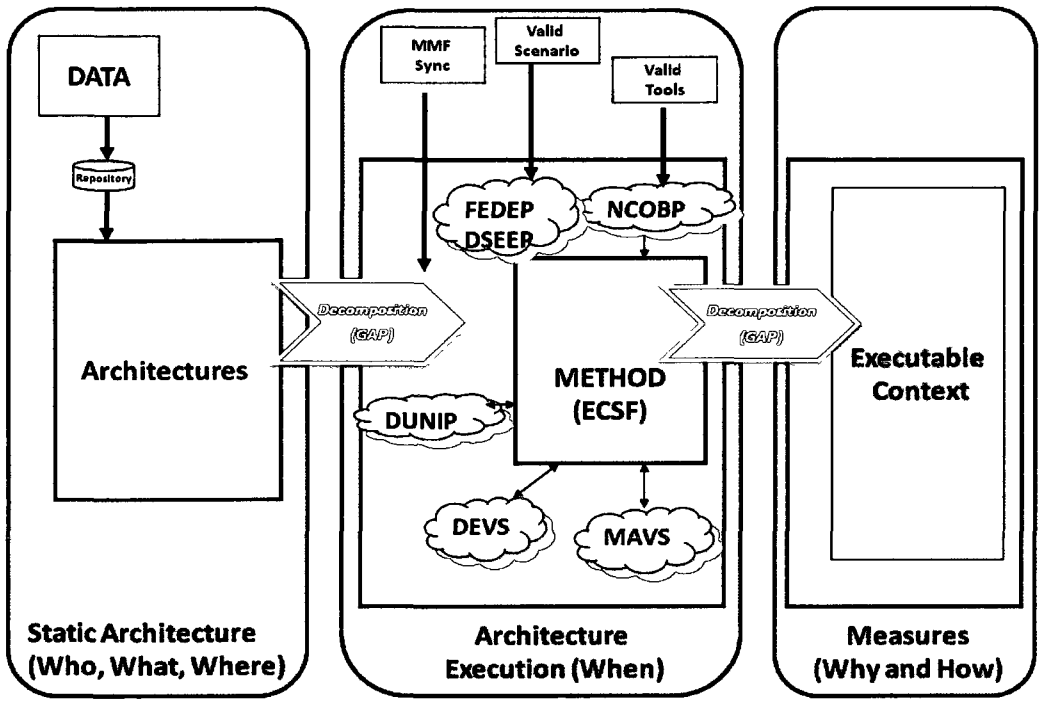


Figure 43: Step 4 – putting it all together

7.3. RESULTS OF THE NEW EXPERIMENT

The key questions that were answered in these experiments were:

1. Does incorporating the "context" affect the evaluation of the system?
2. Does incorporating the "context" lead to different decisions?

This experiment clearly established that the EC method could incorporate systems architectures with an operational context, federated with other models to acquire critical measures of effectiveness and measures of performance relating to system-of-systems capabilities. It was imperative that a full understanding of the "as-is" process be modeled first. Then the "to be" model was developed based on the same context. By analyzing the "as-is" versus "to be" models, EC had the ability to provide quantitative measures as defined by the metrics established in Step 2.

The overall goal is to demonstrate that the EC methodology could assess capabilities and aid in the evaluation of the architecture products into a harmonized method that enables an executable context, as identified in this research, than what has historically been done.

The following qualitative results and recommendations were provided as part of the experiment. The results were based on the following equation:

How:

PSSK (Probability of Single Shot Kill) = PFFR x PE/R x PK/E where:

PFFR = Weapon Free Flight Reliability. PFFR is the probability that all weapon components operate as designed, given a successful aircraft release of a functional weapon. PFFR includes all flight functions after weapon release up to and including warhead detonation. PFFR does not include failures not onboard the weapon.

PE/R = Probability of Engaging the target given a successful release. Given a successful release, PE/R is the probability the weapon acquires, correctly classifies, commits, and guides to the correct target located near the designated aim point. PE/R stresses the seeker and data link and is left open to gives the opportunity to pursue the best solution to achieve the desired PSSK.

PK/E = Probability weapon kills the target given an engagement. PK/E is the probability the weapon achieves the desired kill criteria given successful PFFR and successful PE/R against each individual target set member. Given the above equation the EC method was tailored to qualitatively answer if the architecture could validate the probabilities of a single shot kill of the NEW capability.

Equation 2: PSSK equation

With the results of the "as is" and "to be" models analyzed, the following was gathered and evaluated using the following figure, which represents three different runs.

The key elements of the experiments were the ability to use the data to answer the specific question of PSSK. Below is a synthesis of the results used to determine how EC answered the *how* and the *why*. Figure 44 shows an example of a JCAS model being

executed with a deadlock situation shown in red. The model contains operational process of a targeting Chain for JCAS. Activities requiring resources have couplings to those resources and acquire them during execution. During execution, two threads of execution reach a deadlock situation over resources. The state of the deadlock is two processes that have acquired only one of the two resources needed to execute. Within the deadlock, the blue lines represent resources that have been acquired by an activity, and the orange lines represent resources that have only been requested. The deadlock remains because neither activity will give up its resource. However, logic embedded within the resources is in place to handle prioritizing the competing threads. The logic causes the lower priority thread to yield, allowing the other thread to gain control of its required resources. After the first thread is finished, it releases both resources, and the yielding thread can finally execute. This example shows how the executable context evaluates a theoretical situation like a dead lock in an operational environment.

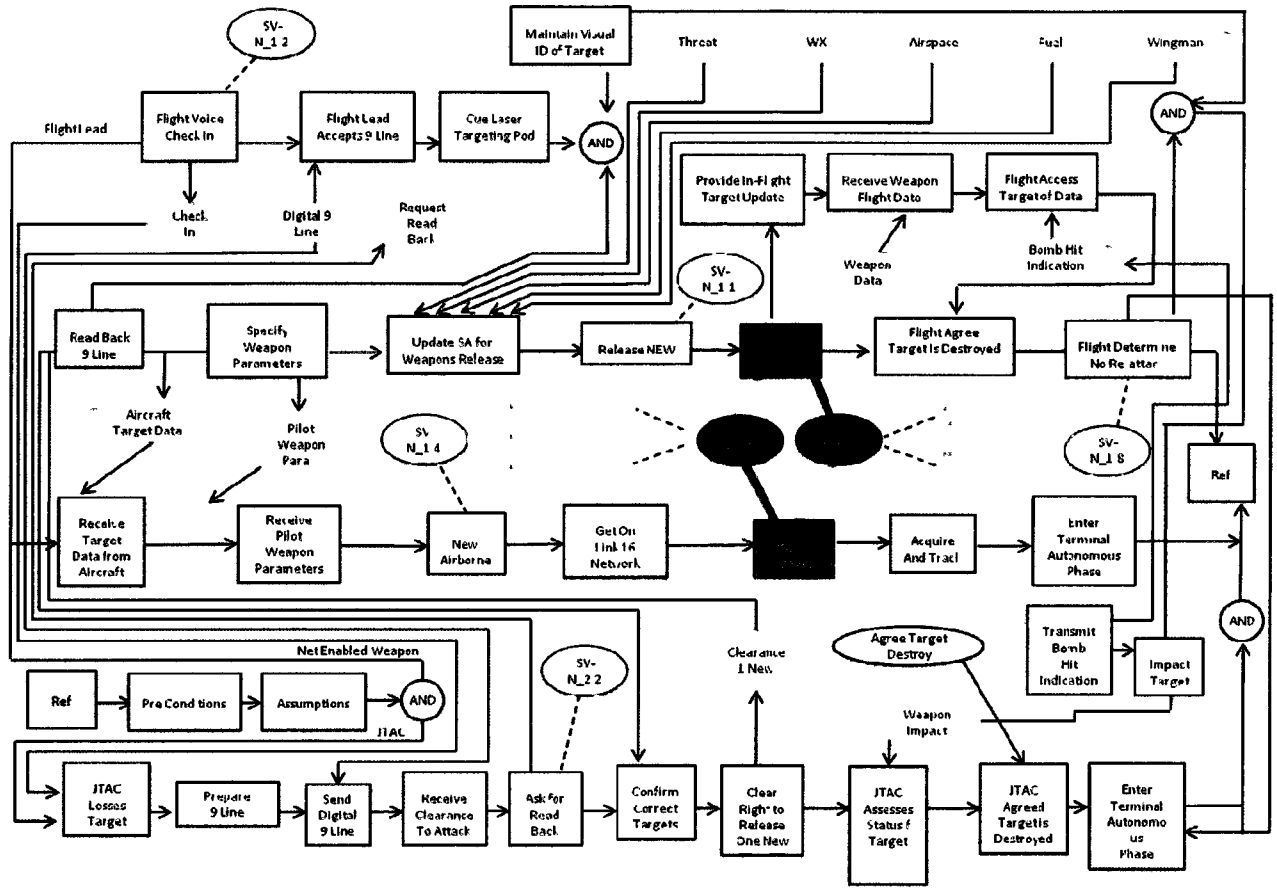
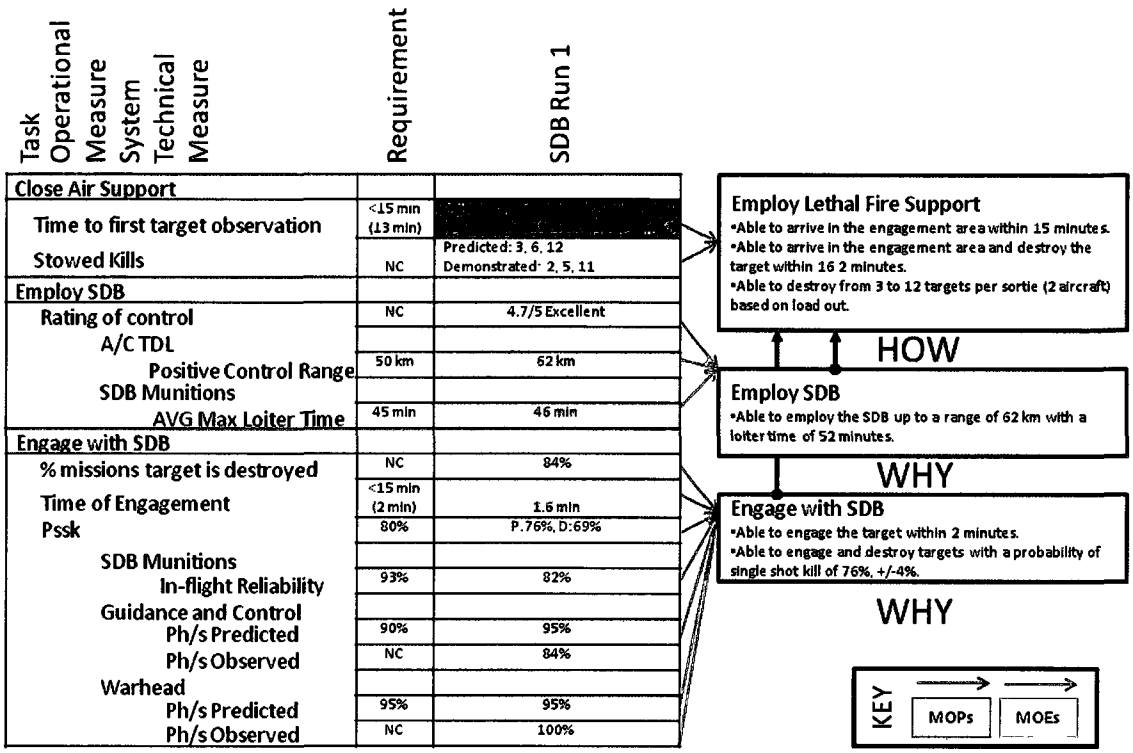


Figure 44: JCAS Dead Lock Model

Figure 45 is the synthesis of run 1 or the Small Diameter Bomb (SDB) that showed "how" to employ the weapon to gather Measures of Effectiveness from the operational activities as shown by the green arrows. The blue arrows show the capture of measures of performance from the systems and answer the "why" to employ and engage. The results are shown to depict MOPs and MOEs.



NC: No defined criteria

Figure 45: "how" and "why" of Run 1 (SDB)

Figure 46 is the synthesis of run 2. This run provided experimental information of a Small Diameter Bomb 2 (SDB 2) conventional system and its existing capabilities. Results provided the following information showing that "how" the SDB 2 employed results and "why" SDB 2 capabilities employed and engaged the target.

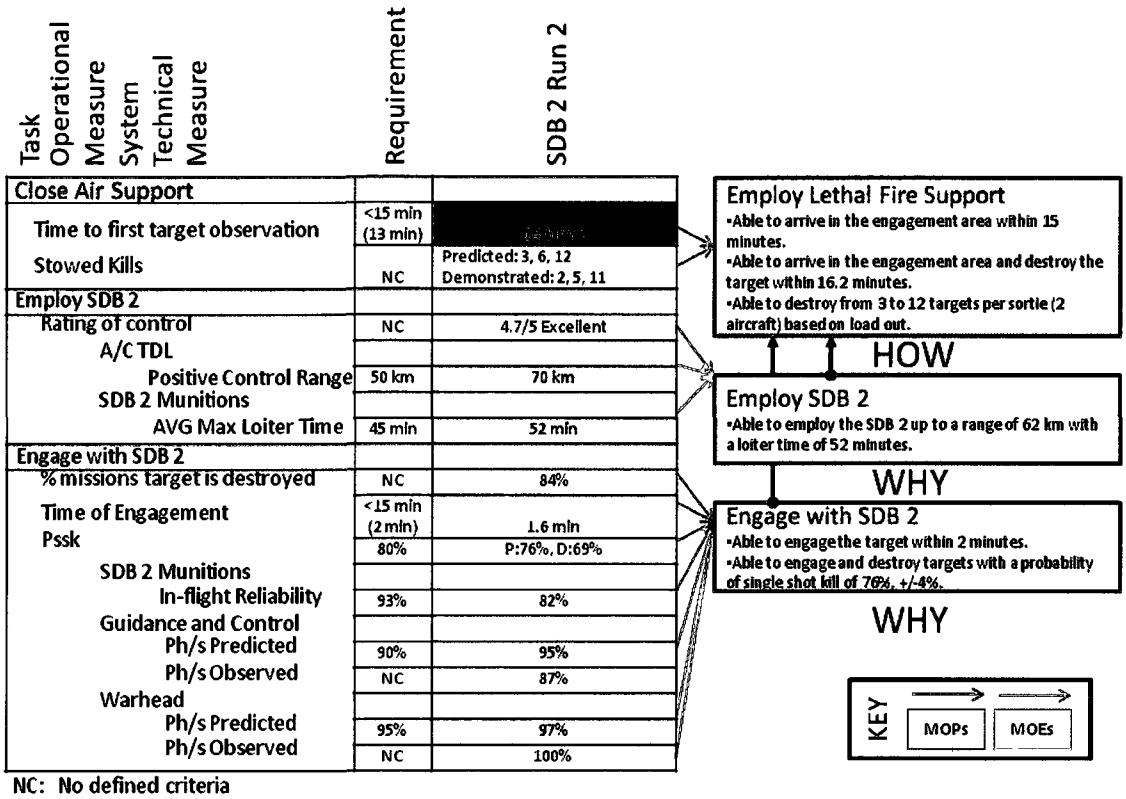


Figure 46: "how" and "why" of run 2 (SDB 2)

Figure 47 is the synthesis of run 3 or the Net Enabled Weapon (NEW). This run took into account "how" to employ the weapon and "why" to employ the weapon and engage the target with experimental and advanced capabilities. Network-enabled weapons will provide the warfighter with the capability to prosecute time sensitive and mobile targets by supplying real-time accurate target information to the weapon from release through impact. In essence, network-centric systems establish communication nodes linking weapons with the most accurate information available. Information will be provided to the weapon by the most timely and accurate source available and not be limited to the delivery platform. In-flight, the weapon receives target location updates and incorporates real-time data into guidance systems for aim point adjustments. This will provide a means to redirect a GPS-guided weapon after release and hold the mobile

target set at risk regardless of weather conditions. If the weapon is equipped with a seeker, the seeker may be preprogrammed to take over and gain greater accuracy for discriminate targets. Overall, the full impact of network-enabled weapons is still unclear, but based on the finding of this research, the benefits clearly touch every element within the kill chain. Networking weapons provides a technological solution that fills a documented capability gap and has the potential to spawn innovation in advanced architectures. More research must be accomplished to properly integrate the architecture without creating stovepipe solutions that meet only near-term needs. To prevent stovepipe solutions and achieve full network weapon integration, an overarching joint portfolio management solution is required to streamline this capability to the warfighter and should be a focus on future research.

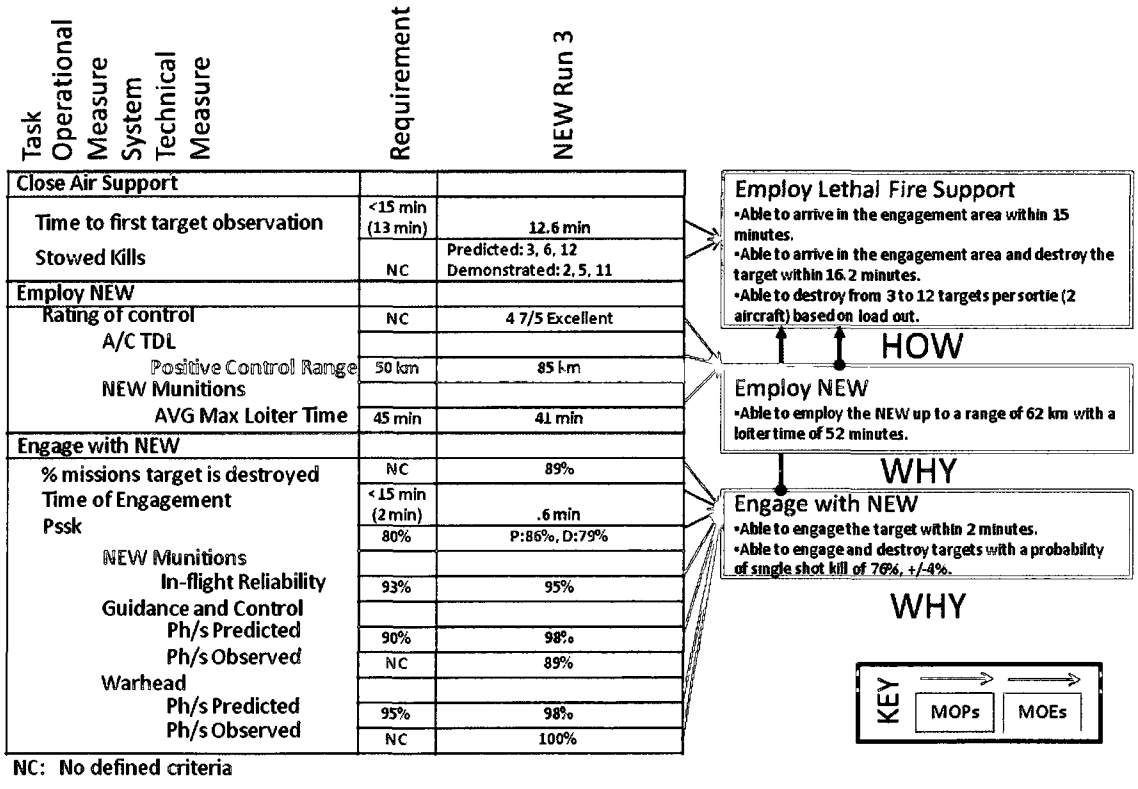


Figure 47: "how" and "why" of run 3 NEW

The results for the runs were analyzed based on the problem formulation established in step 1 of the EC method and the following results were gathered. The key to the analysis on the NEW or "to be "capability is enabled by the following:

1. What are the systems that are affected by this system?
2. What are the systems that affect this system?
3. What environment does this system operate in?
4. Will this system execute within its intended environment as predicted?

The experiment is designed to evaluate the use of a Net-Enabled weapon in comparison to using a conventional weapon. There are significant issues in developing the experimental design. Many of these issues stem from the large scope of the experimental design needed to evaluate the experiment. The experiment needs to correctly and completely describe all of the following items:

- The constraints of the experiment
- The problem and relevant background information
- The defined set of test hypotheses
- The identified variables (independent, dependent, test)
- The tools and techniques
- The preconditions for running the experiment.
- The statistical tests and tools for analyzing the data
- The sources of error
- Ensure that the experiment as a whole is feasible
- Define the number of repetitions
- Define the sources of error
- Define the limitations

Several significant issues have been identified in running the experiment. The first issue concerns the availability of suitable data. In this experiment, each run needs to define the data being used. The third significant issue is meeting the pre-conditions for running each of the experiments. In addition to being a significant amount of work

to develop the tools and materials, the items developed need to be consistent in quality. There are significant issues for developing the experimental design, running the experiment, and analyzing the results. It is important to understand that the issues in running and analyzing the experiment were addressed in the experimental design. The NEW experiment provided a viable method to conduct dynamic, persistent, extensible, measurable, repeatable and interactive testing of processes, architectures and components. With the achievement of this goal, today's challenge of evaluation without an effective method to test and verify tactics, techniques, and procedures (TTP) in an executable context with systems, organizational structure and functions is remedied.

This experiment showed the integration of the independent models into a common method that allows harmonization of system and operational architecture as an executable. This experiment also allows the resulting artifacts to be federated into an executable context showing all external systems and was initialized with an operational scenario that allowed relevant measures of performance on the system and measures of effectiveness on the operational level.

8. CONCLUSION

To conclude, the methodical development and testing of the executable context theory were developed to uphold that this theory is both novel and viable. Results indicate that, by incorporating executable context in system evaluation, context affects the overall evaluation and does lead to different decisions based upon this evaluation. By leveraging literature research and developing an operational and theoretical example, the executable context method exhibited evidence that system-of-systems architecture can be evaluated. This method leverages current executable architecture and architecture analysis methods through modeling and simulation and systems engineering to close the gap and vet executable context as a new method of evaluation.

With today's state-of-the-art in executable architectures, theoretically-sound dynamic analysis of system-of-systems effectiveness and performance is difficult to achieve within an operational environment. The executable context research builds upon and advances the current state-of-the-art in architecture evaluation by simulating operational and system contexts. Figure 48 demonstrates functional orchestration of operational and systems architectures in knowledge-based evaluation of architectures.

Furthermore, this research extended the DUNIP research, enabling an executable federate that allows for operational and systems model transparency and extension of the MMF in a federation of the operational and systems artifacts to ensure that all measures of performance on the system level are computed within the user relevant operational context based on the system's specification and contribute directly to the operational efficiency (MOE) and Systems efficiency (MOP). These extensions enabled operational and system model harmonization for resulting artifacts to be federated into an executable context.

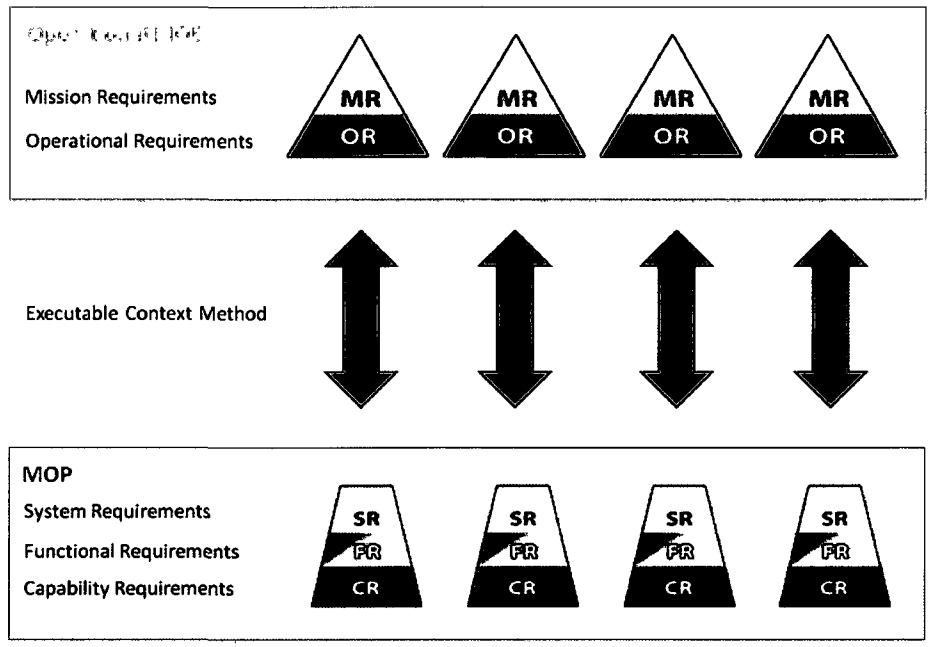


Figure 48: Functional Orchestration of Operational and Systems Architectures using Executable Context Method

With the limited resources in today's economy, organizations must efficiently allocate resources. Executable context research intends to achieve greater evaluation ability with the same or fewer resources. In essence, executable context melds both business and technology best practices to improve efficiency and reduce redundancy by providing higher accuracy in pre-execution evaluation of systems. Executable context models operational and systems architecture specification contributions with a higher degree of accuracy as it considers how an architecture is affected by interacting systems, rather than today's more stovepipe method of system evaluation. This was achieved through extending the DUNIP Specification to enable an executable HLA federate, leveraging and extending the MMF to apply measures to a specific scenario, leveraging the FEDEP and DSEEP processes within in a context and extending the MAVS method to conduct dynamic architecture evaluation.

The executable context method complements other successful approaches such as the Zachman and DoDAF frameworks, agile methods, total quality management, and lean sigma. Further, this research is applicable at multiple system levels – from broad enterprise and business architecture to the intricate systems architecture and technology implementation. Throughout the executable context research, the concept was vetted by analyzing the capability architectures to establish system performance evaluation and effectiveness. The result determined that executable context creates a partial static analysis environment in which static architecture framework products can be more closely examined. This closer examination furthers the study of other routine processes. The current research delivers results that perpetuate repeatable and measurable environments comprised of replaceable components evaluated under different technical, operational and system architectures. The research also led to the creation of a method to convert systems and operational architectures into an executable context.

Based upon these findings, executable context may also provide added value in service orientation, business processes, and information and business rules (limited by certain problems and opportunities within these). Evidence presented throughout this research indicates that systems that utilize architected solutions out-perform chaotic systems when architecture is applied at the right levels. Executable context methods are also developed to determine when an architected solution is appropriate and how much detail is required. These methods also provide further insight to determine how to design a plan or processor to convey information effectively and efficiently without stifling creativity and diversity.

Finally, executable context research derives information and generalizes architecture frameworks such as DoDAF, Zachman, 4+1 view and others to evaluate that the current state-of-the-art in executable architectures are not able to analyze operational system-of-systems in a dynamic fashion, as these foundational principles evaluate the architectures within their own operational environment. The extensible

method developed within this research is an evolution of these concepts that is designed to close this gap through considering systems within their contexts.

9. EXTENSIBILITY OF THE RESEARCH

Many of today's engineering and architecture environments have adopted a system-of-systems approach to developing and sustaining their capabilities. The focus has shifted from platform-based programs to integrated capability-based development, management and sustainment programs. The EC method supports strategic planners to define the long-range capability roadmaps, evaluate program requirements and determine spending priorities. While traditional systems architectures focus on the operational systems and technical standards, it is the mission element of portfolio management that has increasingly drawn attention. Capability managers struggle with how to best represent the intended mission element of a capability within the system-of-systems construct.

Continuous improvement and success of portfolio management ensure that the appropriate resources are allocated to their authorized portfolio components. Since portfolios rely on projects in order to achieve their strategic goals, they are interconnected and the improvement and success of portfolios has direct influence on the success of system-of-systems projects. This includes:

- System-of-systems acquisition management - a significant increase in complexity over traditional system acquisition.
- Developments that require significant numbers of technologies be integrated to one another.
- Challenges to traditional development monitoring tools and cost models.
- Need to capture integration complexity.
- Level of effort required to connect individual components.
- Unintended consequences - high degree of inter-linkage between components can cause unintended impacts to overall system performance.
- Components are modified from original use.
- Technology change: replaced throughout the system life.

When DoDAF architectures represent test events, a capability is needed for monitoring, visualizing and analyzing events from a DoDAF perspective. Analysts familiar with DoDAF are able to examine how various lower-level messages and events captured during a distributed test environment relate to both system and operational architecture viewpoints. Analysis and measures calculated during events must also be overlaid onto detailed visualizations from a DoDAF perspective. Following the release of DoDAF 2.0, the capability also attempts to follow the DoDAF Meta-model (DM2) and the idea of enabling users with the flexibility to construct hybrid views. Given that the data model is implemented as ontology, the capability also explores the ability of revealing inferred relationships using reasoning engines. Given the ability to reveal additional relationships based on those explicitly defined, the capability attempts to convey unexpected and possibly critical information to users.

This research describes a prototype that implements a data model-based on DM2 and the services that produce meaningful executable context for the Net Ready Key Performance Perimeters which is comprised of the following elements: compliance with the Net-Centric Operations and Warfare (NCOW) Reference Model (RM), applicable Global Information Grid (GIG) Key Interface Profiles (KIP), DOD information assurance requirements, and supporting integrated architecture products required to assess information exchange and use for a given capability.

Extension of the research method developed in this thesis has the ability to provide an integrated approach to system-of-systems evaluation or the ability to evaluate more than one system at a time. It uniquely leverages the principles of resource issues for capability development and management, specifically in the key areas of personnel, resources and activities. The gap between existing "as is" capability and the desirable "to be" capability require a measure for evaluating key incremental capability of the "to be" state. EC has the potential to provide value-focused metrics, including system-level measures of performance and context-based measures of effectiveness, which could lead to integrated capability metrics suitable to link enterprise strategic guidance to an engineered capability portfolio. EC could enable an

analytical approach based on a simulation-based environment; client capability engineering teams understand the impact of alternative capabilities on variables such as strategic key performance parameters, operational and system performance, lifecycle costing, personnel training requirements and methods. This research may be used to implement an evaluation process in which needs and resources are integrated early and resources are committed incrementally based on the achievement of specific levels of knowledge at established decision points.

This research may also be used to prioritize programs based on the relative costs, benefits and risks of each investment to ensure a balanced portfolio. EC could be used to require increasingly precise cost, schedule and performance information for each alternative that meets specified levels of confidence and allowable deviations at each decision point leading up to the start of product development. Further, EC may enable empowered portfolio managers to prioritize needs, make early go/no-go decisions about alternative solutions and allocate resources within fiscal constraints. Finally, EC could provide a much needed approach to optimization of operational and systems requirements.

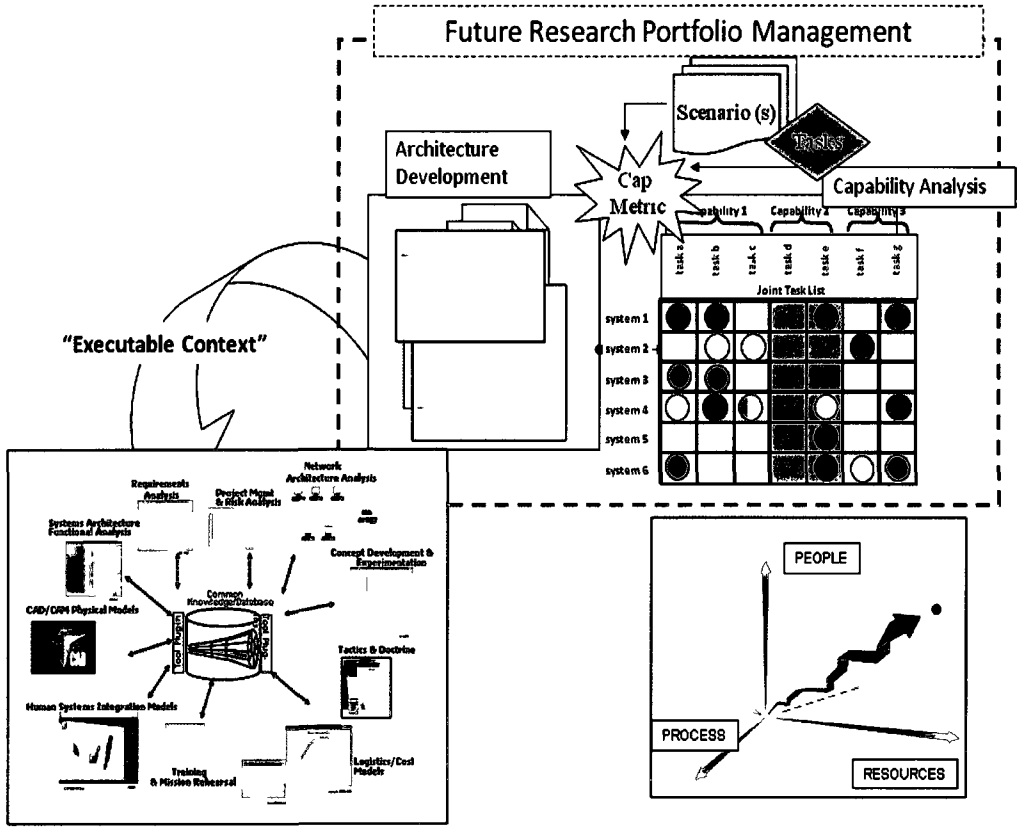


Figure 49: Future research – executable context for portfolio management

10. REFERENCES

Anderson, James H. & Kim, Yong-jik. (2001). *Shared-memory mutual exclusion: Major research trends since 1986.*

Architecture Reviews: Practice and Experience by Joseph F. Maranzano, Sandra A. Rozsypal, Gus H. Zimmerman, Guy W. Warnken, Patricia E. Wirth, and David M. Weiss IEEE SOFTWARE March/April 2005, pages 34-43.

Arthur D. Hall (1962). *A Methodology for Systems Engineering*. Van Nostrand Reinhold. ISBN 0442030460.

Atkinson, K. (2004). *Modeling and Simulation Foundation for Capabilities-Based Planning*. Spring Simulation Interoperability Workshop. p. 8.

Austin, D., Barbir, A., Ferris, C., Garg, S. (2004). *Web Services Architecture Requirements.* W3C Working Group Note.

B. P. Zeigler, 2003DEVS Today: Recent Advances in Discrete Event based Information Technology, MASCOTS Conference.

Balci, O. (1987) Credibility *and Assessment of Simulation Results: The State of the Art.* Proceedings of the Conference of Methodology and Validation, pp.6-9.

Banks, J., Gerstein, S. & Searles, S.P. (1987) *Modeling, Processes, Validation and Verification of Complex Simulations: A Survey.* Proceedings of the Conference on Methodology and Validation pp. 13-18.

Barbar, M.A., et. al. (2004) *A Framework for Classifying and Comparing Software Architecture Evaluation Methods*, Proceedings from the 2004 Australian Software Engineering Conference. pp. 309-318.

Bar-Yam Y., Allison, M., Batdorf, R., Chen, H., Generazio, H., Singh, H., & Tucker, S. (2004). *The Characteristics and Emerging Behaviors of System of Systems System-*

of-systems. NECSI: Complex Physical, Biological and Social Systems Project. http://necsi.org/education/oneweek/winter05/NECSISoS.pdf. Ref. May 2007.

Bass, L., Clements, P., & Kazman, R. (1998). *Software Architecture in Practice*, Addison-Wesley.

Best Current Practices: Software Architecture Validation. (1990). AT&T Bell Labs.

Bredemeyer, Dana & Ruth Malan, "The Role of the Architect," white paper published on the Resources for Software Architects 2004.

Bredemeyer, Dana. (1999). James *Madison and the Role of the Architect.* http://www.bredemeyer.com/papers.htm.

Bredemeyer, Dana. (2007). James *Madison and the Role of the Architect.* http://www.bredemeyer.com/papers.htm.

Brown, Allen. (2009). *An Introduction to Model Driven Architecture.* IBM. http://www.ibm.com/developerworks/rational/library/3100.html.

Buede, Dennis M. (2000). *The Engineering Design of Systems: Models and Methods.* John Wiley & Sons, Inc. p. 38.

Carlock, P.G. & Fenton R. E. (2001), *System-of-Systems (SoS) Enterprise Systems for Information-Intensive Organizations*. Systems Engineering, Vol. 4, Issue 4. pp. 242-261.

Clements, Paul. 1996. A Survey of Architecture Description Languages. In *Proceedings of the 8th International Workshop on Software Specification and Design* (IWSSD '96). IEEE Computer Society, Washington, DC, USA, 16-25.

Chairman, JCS Instruction 3170.01D. (2004). Joint Capabilities Integration and Development System.

Chairman, JCS Instruction 6212.01D. 2006. Interoperability and Supportability of Information Technology and National Security Systems.

Charles, Philipp, Turner, Phil. (2004) Capabilities based acquisition ... from theory to reality. CHIPS.

Charles Keating, Ralph Rogers, Resit Unal, David Dryer, et al. "System of Systems Engineering," Engineering Management Journal, Vol. 15, no. 3, pp. 36.

Charles Keating, "Research Foundations for System of Systems Engineering," 2005 IEEE International Conference on Systems, Man and Cybernetics, Waikoloa, Hawaii, October 10–12, 2005. pp. 2720–2725.

Crnkovic, Ivica & Larsson Magnus. (June 2004). *Classification of quality attributes for predictability in component-based systems*. IEEE DSN 2004 Workshop on Architecting Dependable Systems; Florence, Italy.

Defense Modeling and Simulation Office (DMSO). (1997). *DoD Modeling and Simulation Glossary*. Alexandria, VA.

Deitz, P.H., Sheehan, J.H., Harris, B.A., Wong, A.B.H., Bray, B.E., Purdy, E.M. *The Military Missions and Means Framework (MMF)*. 2005

DOD Architecture Framework, V1.0, Vol. I and II, 15 (August 2003).

DoD Instruction 5000.2. (2003). Operation of the Defense Acquisition System.

DoD Metadata Registry and Clearinghouse. http://diides.ncr.disa.mil/mdregHomePage/mdregHome.portal/. Accessed Feb 2010.

DoDAF Working Group. (2003). *DOD Architecture Framework Ver. 1.0 Vol. III*: Deskbook, DOD.

DoD Net-Centric Services Strategy, Strategy for a Net-Centric, Service Oriented DoD Enterprise, March, (2007), issued by ASD (NII)/DoD CIO.)

Else, Steven. (2005). *As-is, Could Be and Possible Transition Consideration*, Center for Government Transformation.

Forsberg, K., Mooz, H., Cotterman, H. *Visualizing Project Management,* 3rd edition, John Wiley and Sons, New York, NY, 2005. Pages 108-116, 242-248, 341-360.

Garcia, Johnny, Browning, J. (2006). Innovations in Process Modeling as Applied to JFCOM Joint Experimentation Directorate Division Experiment Management Teams. Spring SIW 06S-SIW-015.

Garcia, Johnny. (2009). *Executable Architecture Analysis Model (EAAM).* Proceedings of the ITEA LVC Conference Las Cruses, NM.

Garcia, Johnny. (2010). *Methodology Supporting Architecture Evaluations (MAVS).* Spring Simulation Conference.

Gideon, J. & Dagli, C. (2005). *Taxonomy of System-of-Systems*. CD Proceedings of CSER Conference on Systems Engineering Research. Hoboken, NJ.

Greene, J.C. (1987). *Uses and misuses of mixed-method evaluation designs*. Proposal for the 1988 annual meeting of the American Education Research Association (New Orleans). p. 22.

Handley, Holly, Zaidi, Zainab, Levis, Alexander. (2000). *The Use of Simulation Models in Model Driven Experimentation, System Architectures Laboratory.* C3I Center, George Mason University.

INCOSE Systems Engineering Handbook V.3. (2006). INCOSE.

International Council On Systems Engineering (INCOSE), *Systems Engineering Handbook Version 3.1,* August 2007, pages 3.3 to 3.8

Institute of Electrical and Electronics Engineers. (2000). IEEE Stat 1471-2000: IEEE Recommended Practice for Architectural Description of Software-Intensive Systems.

Institute of Electrical and Electronics Engineers. (2011). IEEE Stat 1730-2010: IEEE Recommended Practice for Distributed Simulation Engineering and Execution Process (DSEEP).

International Organization for Standardization. (2002). *ISO/IEC 15939:2002 Software engineering – Software Measurement Process.* Geneva, Switzerland: Author.

Joint Publication 1-02, Appendix A-1, p.333, 3/2007

Kruchten, Philippe. (1995). *The 4+1 View Model of Architecture*, IEEE Software, vol. 12, no. 6, pp. 42-50.

Lee, J., Choi, M., Jang, J., Park, Y., Jang J., Ko, B. (2005). *The Integrated Executable Architecture Model Development by Congruent Process, Method, and Tools.* Proceedings of the 2005 Command and Control Research and Technology Symposium, CCRTS '05, Tyson's Corner, VA, p. 259.

Levis, A. (1993). *Proceedings of the Symposium on Command and Control Research.* National Defense University, Fort Lesley J. McNair, Washington D.C.

Levis, Alexander & Wagenhals, Lee. (2000). *C4ISR Architectures I: Developing a Process for C4ISR Architecture Design*. Systems Engineering, Vol. 3, No. 4, pp. 225-247.

Levis, Alexander & Wagenhals, Lee. (2000). *C4ISR Architectures: I. Developing a Process for C4ISR Architecture Design*, System Architectures Laboratory, C3I Center, MSN 4D2, George Mason University.

Levis, Alexander & Wagenhals, Lee. (2006). *DoD Architecture Framework Implementation.* Class notes published by AFCEA, Fairfax, VA.

Levis, Alexander, National Missile Defense (NMD) Command and Control Methodology Development. George Mason University, Fairfax, VA. 1993.

Frederic D. Mckenzie, Mikel D. Petty, and Qingwen Xu. 2004. Usefulness of Software Architecture Description Languages for Modeling and Analysis of Federates and Federation Architectures. *Simulation* 80 (11): 559-576.

Maier, M. W. (2005). *Research Challenges of System-of-Systems*. IEEE International Conference on Systems Man and Cybernetics, Vol. 4. pp. 3149-3154.

Marazano, Joseph F.; Rozsypal, Sandra A.; Zimmerman, Gus H.; Warnken, Guy W.; Wirth, Patricia E.; Weiss, David M. (2005). *Architecture Reviews: Practice and Experience*. IEEE SOFTWARE March/April Edition. pp. 34-43.

Missile Defense Agency - Department of Defense Documentation of Verification, Validation & Accreditation (VV&A) for Models and Simulations. (2008) Missile Defense Agency.

Stephen J. Mellor, Marc Balcer. 2002. Executable UML: A Foundation for Model-Driven Architectures. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

Merriam-Webster's Collegiate Dictionary (11th ed.). (2003). Springfield, MA: Merriam-Webster.

Mittal, Saurabh, Mak, E. Nutaro, J.J. (2007). DEVS-Based Dynamic Modeling & Simulation Reconfiguration using Enhanced DoDAF Design Process. Special issue on DoDAF.

Mittal, Saurabh, Risco-Martin, J.L. & Zeigler, Bernard P. (2007). *DEVSML: Automating DEVS Execution Over SOA Towards Transparent Simulators.* Special Session on DEVS Collaborative Execution and Systems Modeling Over SOA, DEVS Integrative M&S Symposium DEVS '07, Spring Simulation Multi-Conference.

Mittal, Saurabh. (2006). *Extending DoDAF to Allow DEVS-based Modeling and Simulation.* Special issue: DoDAF Journal of Defense Modeling and Simulation JDMS, Vol. III, No. 2.

Mittal, Saurabh. (2007). DEVS Unified Process for Integrated Development and Testing of Service Oriented Architectures. PhD Dissertation, University of Arizona.

Mogul, Jeffrey C., Ramakrishnan, K. K. (1996). *Eliminating receive livelock in an interrupt-driven kernel.* http://citeseer.ist.psu.edu/326777.html.

NATO Code of Best Practice for C2 Assessment. (2002) Command and Control Research Program (CCRP).

Object Management Group. (2009). *Model Driven Architecture: The Architecture of Choice for a Changing World.* http://www.omg.org/mda/executive_overview.htm. Ref: January 2009.

Pawlowski, T., Barr, P., Ring, S., Segarra, S. (2004). *Executable Architecture Methodology for Analysis, FY04 Report.* MITRE Technical Report 05W0000034.

Polanyi, M. & Prosch, H. (1998). *Meaning, Chicago.* The University of Chicago Press Delphi KM Conference, San Diego, CA.

Popper, S., Bankes, S., Callaway, R., & DeLaruentis, D., *System-of-Systems Symposium: Report on a Summer Conversation.* July 21-22, 2004, Potomac institute for Policy Studies, Arlington, VA.

Rechtin, E. (1991). Systems Architecting: Creating and Building Complex Systems. Prentice-Hall.

Rensselaer, Dave Hollinger. (2004). Lecture notes. http://www.cs.rpi.edu/academics/courses/fall04/os/c10/index.html.

Rensselaer, Dave Hollinger. (2005). Lecture notes. http://www.cs.rpi.edu/~hollingd/opsys-spring2005/notes/Chapter3/Chapter3.pdf .

Sage, A. P. & Rouse, W. B. (Eds.). (1999), *Handbook of Systems Engineering and Management*. John Wiley and Sons. p. 264.

Sage, A. P. & Cuppan, C.D. (2001). *On the Systems Engineering and Management of Systems-of-Systems and Federations of Systems*. Information, Knowledge, Systems Management, Vol. 2, Issue 4, pp. 325-45.

Schlager, J. (1956). *Systems engineering: key to modern development.* IRE Transactions EM-3: 64–66. doi:10.1109/IRET-EM.1956.5007383.

Schmidt, D.C. (February 2006). "Model-Driven Engineering". *IEEE Computer* 39 (2). Retrieved 2011-02-16.,

Secretary of the Navy (SECNAV) Instruction 5200.40. (1999). *Verification, Validation, and Accreditation (VV&A) of Models and Simulations.* http://navmsmo.hq.navy.mil/policy/directives. Accessed February 2010.

Seliger, R. (1997). An Approach to Architecting Enterprise Solutions. HP Journal.

Sirer, Emin Gun. (2001). *Deadlock.* http://www.cs.cornell.edu/courses/cs414/2003sp/lectures/9-deadlock.pdf.

Systems Engineering for Intelligent Transportation Systems". US Dept. of Transportation. p. 10. http://ops.fhwa.dot.gov/publications/seitsguide/seguide.pdf. Retrieved 2011-02-09.

Systems Engineering Handbook, version 2a. (2004). INCOSE.

Taguchi, Genichi. *Introduction to Quality Engineering,* "The Development of Quality Engineering." *ASI Journal* 1, no. 1. White Plains, NY: Asian Productivity Organization, UNIPUB. pp. 1-4. 1991.

Tang, A., Han, J., Chen, P. (2004). SUTIT-TR2004.01 – A Comparative Analysis of Architecture Frameworks.

Venkatesh, S., Smith, J. Deuermeyer, B., Curry, G. (2000). *Deadlock Detection and Resolution for Discrete Event Simulation: Mutiple-Unit Seizes*. Texas A&M University. http://tamcam.tamu.edu/papers/multirev/multirev.htm.

Weiss, Elliott N. (2009). Brief Introduction to Taguchi Methods.

Youngblood, S. M. & Pace, D. K. (1995). *An Overview of Model and Simulation Verification, Validation, and Accreditation.* Johns Hopkins APL Technical Digest 16 (2). p. 197-206.

Zachman, John A. (1987). *A framework for information systems architecture*. IBM Systems Journal. Vol 26. No. 3. http://www.zachmaninternational.com/images/stories/ibmsj2603e.pdf.

Zachman, John A. (1997). Concepts of the Framework for Enterprise Architecture: Background, Description and Utility. Zachman International. Accessed Jan 2009.

Zeigler, B.P. (2003). DEVS Today: Recent Advances in Discrete Event based Information Technology, MASCOTS Conference.

Zinn, Andrew W. (2004). *The Use of Integrated Architectures to Support Agent-Based Simulation: An Initial Investigation*. Air Force Institute of Technology, Graduate School of Engineering and Management, Thesis.

Zöbel, Dieter. (1983). *The Deadlock problem: a classifying bibliography*. ACM SIGOPS Operating Systems Review 17 (4): 6–15. ISSN 0163-5980 http://doi.acm.org/10.1145/850752.850753.

11. APPENDIX A CODE BLOCK FOR CLASS

```
// This class has been trimmed for brevity.

// Some global variable declarations and helper functions have been omitted for brevity.

public class OV5_View extends ViewableDigraph {

  public OV5_View(String name, QUEUE_InternalEvent internalEventQueue){

    super(name);

    // Generate DEVS models for all models connected this Operational Activities (Systems, Functions, Resources, etc...)

    LIST_NODE ConnectedModels = SINGLETON_DoDAF_Models.getInstance()._All_Connected_Models;

      for(int i = 0; i < ConnectedModels._size; i++)

    {

      // Each executable architecture data type has a method that generates its DEVS model

      NODE model = (NODE) ConnectedModels.getNodeByIndex(i);

      model.generateDEVS(internalEventQueue);

    }

      // Generate DEVS models for "Operational Activities"

    // Generate DEVS ports to other models based on connectivity

    // The architecture is currently stored in the data model

      LIST_NODE OV5_Nodes = SINGLETON_DoDAF_Models.getInstance()._All_OV5_Nodes;

      for(int i = 0; i < OV5_Nodes._size; i++)

    {

      NODE_OV5 ov5 = (NODE_OV5) OV5_Nodes.getNodeByIndex(i);
```

```
    ov5.generateDEVS(_internalEventQueue);
    DEVS_OV5 devs_ov5 = ov5.getDEVS_Model();
            LIST_NODE ConnectedModels = ov5._ConnectedModels;
            for(int j = 0; j < ConnectedModels._size; j++)
    {
      NODE model = (NODE) ConnectedModels.getNodeByIndex(j);
      String ModelName = model.getName();
        DEVS_MODEL devs_model = model.getDEVS_Model();
      // Add dynamic ports to other models
      devs_ov5.addOutport("status_" + ModelName);
      devs_ov5.addOutport("done_" + ModelName);
      devs_model.addOutport("status_" + ov5._NodeName);
       }
            devs_ov5.initialize();
    add(devs_ov5);
  }
      // Officially add and initialize other connected DEVS models
for(int i = 0; i < ConnectedModels._size; i++)
{
    NODE model = (NODE) ConnectedModels.getNodeByIndex(i);
           DEVS_MODEL devs_model = node.getDEVS_Model();
    devs_model.initialize();
    add(devs_model);
}
```

```
                // Make connections (DEVS couplings) between Operational Activities
and other models
            for(int i = 0; i < OV5_Nodes._size; i++)
            {
        NODE_OV5 ov5 = (NODE_OV5) OV5_Nodes.getNodeByIndex(i);
 LIST_NODE ConnectedModels = ov5._ConnectedModels;


              for(int j = 0; j < ConnectedModels._size; j++)
              {
                 NODE model = (NODE) ConnectedModels.getNodeByIndex(j);
                 String ModelName = model.getName();
                 // OV5 -> Supportive Model (status, done)
                 addCoupling(ov5.getDEVS_Model(), "status_" + ModelName,
model.getDEVS_Model(), "request");
                 addCoupling(ov5.getDEVS_Model(), "done_" + ModelName,
model.getDEVS_Model(), "done");
              }
            }
                 showState();
          }
        }
```

12. APPENDIX B DEVS JAVA CODE BLOCK

```
// This class has been trimmed for brevity, but it does contain all the major functions present in all DEVS models in DEVSJAVA.

// Some global variable declarations and helper functions have been omitted for brevity.

public class DEVS_OV5 extends ViewableAtomic {

    public DEVS_OV5(String name, int NodeID, QUEUE_InternalEvent internalEventQueue) {

        super(name);

            // Some standard input and output ports for this model

        addInport("in");

        addInport("CommComplete");

            // Other Outputs are autogenerated;

        addOutport("out");

        addOutport("null");

    }

    public void initialize() {

        phase = "passive";

        sigma = INFINITY;

        super.initialize();

    }

    // This external transition function executes when the model receives an external message

    // Messages could be empty triggers (from a previous Operational Activity) or state

    // updates from other models (Resources, Mediators, Communications model)
```

```
public void  deltext(double e, message x)
{
  // The function acts differently depending on the its current state
  // Is this the start of the invocation
  if (phaseIs("passive"))
  {
     // Parse the incoming message
    for (int i=0; i< x.getLength();i++)
    {
        // Also figure out at which "port" the message arrived.  DEVS models can have multiple input and output ports
      if (messageOnPort(x,"in",i))
      {
        entity val = x.getValOnPort("in", i);
        Universal_Message message = (Universal_Message)val;
        int messageType = message._messageType;
          // Messages also have multiple types
          // This function can also act differently depending on the port where a message arrived
        if(messageType == MAIN_Lookup.UNIVERSAL_MESSAGE_START)
        {
          _currentState = MAIN_Lookup.DEVS_BUSY;
          _timeStarted = SINGLETON_Time.getInstance().getCurrentTime();

            // Does this model send an HLA JSAF order immediately
```

```
                    // "_internalEventQueue" is a global event queue used by any
component that generates events
                    // The "Event" object contains the logic that actually sends the HLA
order
                    if(JSAF_Order)
                    {
                      _internalEventQueue.enqueue(new
EVENT_JSAF_Order(_nodeID._orders));
                    }

                    _internalEventQueue.enqueue(new
EVENT_DB_StateChange(_NodeID, _name, _currentState));
                      // Check if this model require a transmission to be carried out
by the Communications model
                      // "holdIn" statements put the model in a certain state
                    if(!requireTransmission())
                    {
                      holdIn("ToConnectedModels",0);
                    } else {
                      holdIn("ToComms",0);
                     }
                  }
                }
              } else if(phaseIs("awaitingComms")) {
                for (int i=0; i< x.getLength();i++)
```

```
            {
                if (messageOnPort(x,"CommComplete",i)) {
                    entity val = x.getValOnPort("CommComplete", i);
                    Universal_Message message = (Universal_Message)val;
                    _internalEventQueue.enqueue(new
EVENT_DB_StateChange(_NodeID, _timeStepStarted + duration, duration));
                    holdIn("done", 0);
                }
            }
        }
    }

    // This function executes at necessary intervals for the model
    // The function behaves differently depending on the current state
    public void  deltint( )
    {
      if(phaseIs("passive"))
      {
        holdIn("passive",INFINITY);
       } else if(phaseIs("ToConnectedModels")) {
        holdIn("Delaying",_calculation);
         } else if(phaseIs("Delaying")) {
       if(!_requiresTransmission())
       {
```

```
            holdIn("ToComms",0);
        } else {
    ExecuteAdditionalLogic();
            holdIn("Delaying",_calculation);
        }
            } else if(phaseIs("ToComms")) {
         holdIn("awaitingComms",INFINITY);
        } else if(phaseIs("done")) {
        holdIn("passive",INFINITY);
      }
    }

    // This function executes when an internal and external transition occur at the
same time
    public void deltcon(double e, message x)
    {
      deltint();
      deltext(0,x);
    }
    // This function executes when the model produces outputs
    public message out()
    {
       message m = new message();
       if(phaseIs("passive"))
```

```
            {
              Universal_Message value = new Universal_Message();
              content con = makeContent("null", value);
              m.add(con);
            } else if(phaseIs("ToConnectedModels")) {
              // This code block has prepare messages to send to all other models that
this Operational Activity is connected to
              Universal_Message value = new Universal_Message();
               for(int i = 0; i < _connectedModels._size; i++)
              {
                String ModelName = _connectedModels.getNodeByIndex(i).getName();
                value._messageType                                              =
MAIN_Lookup.UNIVERSAL_MESSAGE_TYPE_REQUEST;
                value._sendingModel = _name;
                value._destinationModel = ModelName ;
                content con = makeContent("status_" + ModelName , value);
                m.add(con);
              }

            } else if(phaseIs("ToComms")) {
              // This code block prepares messages for the Communications model
               // Associated System Content for COMMS model
               Universal_Message value = new Universal_Message();
              value._messageType                                                =
MAIN_Lookup.UNIVERSAL_MESSAGE_TYPE_REQUEST;
```

```
                value._sendingModel = _name;
                value._destinationModel = _COMMS_Model;
                content con = makeContent("status_" + _COMMS_Model._name, value);
                m.add(con);
                // Send Message to COMMS model for transmission simulation
                _internalEventQueue.enqueue(new
COMMS_EVENT_Transmission_START(_transmission._sender.getObjectID(),
                    _transmission._receiver.getObjectID(),50,    _name,    Protocols.TCP,
_NodeID));
            } else if(phaseIs("awaitingCOMMs")) {
              Universal_Message value = new Universal_Message();
              content con = makeContent("null", value);
              m.add(con);
            } else if(phaseIs("done")) {
              // Notify Connected Models that this Operational Activity is finished
              for(int i = 0; i < _ConnectedModels._size; i++)
              {
                Universal_Message value = new Universal_Message();
                String ModelName = _ConnectedModels.getNodeByIndex(i).getName();
                value._messageType                                                    =
MAIN_Lookup.UNIVERSAL_MESSAGE_TYPE_SYSTEM_DONE;
                value._sendingModel = _name;
                value._destinationModel = ModelName;
                content con = makeContent("done_" + ModelName, value);
                m.add(con);
```

```
            }
            // Trigger the next Operational Activity(ies) via a message
            Universal_Message value = new Universal_Message();
              value._messageType                                    =
MAIN_Lookup.UNIVERSAL_MESSAGE_TYPE_DONE;
            value._sendingModel = _name;
            value._destinationModel = _nextOV5;
            content con = makeContent("out", value);
            m.add(con);
          } else if(phaseIs("Delaying")) {
           Universal_Message value = new Universal_Message();
            content con = makeContent("null", value);
            m.add(con);
             } else {
            Universal_Message value = new Universal_Message();
            content con = makeContent("null", value);
            m.add(con);
          }
        return m;
          }
```

13. VITAE

Johnny J. Garcia

Dr. Garcia is Founder and CEO of one of the fastest growing Modeling and Simulation (M&S) and Information Security Company in Virginia - SimIS Inc. He has engineering experience that includes systems architecture design, software development, database development, C4I systems development, logistics systems development, and new technology insertion for the Department of Defense, Department of Energy, National Aeronautics and Space Administration (NASA), Department of Commerce and Department of Homeland Security. Dr Garcia received his Bachelors of Arts and Bachelors of Science Degrees from St Leo College, His Masters of Business Administration (MBA) and Masters of Science (MS) from Florida Institute of Technology and his Ph.D. in Modeling and Simulation from Old Dominion University. Dr Garcia is a veteran of the US Navy and a member of Simulation Interoperability Standards Organization (SISO), Institute of Electrical and Electronics Engineers (IEEE), International Council of Systems Engineering (INCOSE). He attends Ascension Catholic Church in Virginia Beach and a member of Young Life and Cursillo groups of Tidewater and is the proud father of wonderful twin daughters Hope and Faith and is married to his lovely wife Lorena. Dr Garcia is a leader in the community and has been recognized as an expert in M&S throughout the country and the world.