THESIS

ENGINEERING SOFTWARE FOR INTEROPERABILITY THROUGH USE OF ENTERPRISE ARCHITECTURE TECHNIQUES

by

Jennifer L. Parenti

March 2003

Thesis Advisor:	Valdiz Berzins
Second Reader:	Richard Riehle

REPORT DOCUMENTATION PAGE			*Form Approved OMB No. 0704-0188*

1. AGENCY USE ONLY ***(Leave blank)***	**2. REPORT DATE** March 2003	**3. REPORT TYPE AND DATES COVERED** Master's Thesis

4. TITLE AND SUBTITLE **Engineering Software for Interoperability through Use of Enterprise Architecture Techniques**	**5. FUNDING NUMBERS**
6. AUTHOR(S) Jennifer L. Parenti	
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) Naval Postgraduate School Monterey, CA 93943-5000	**8. PERFORMING ORGANIZATION REPORT NUMBER**
9. SPONSORING / MONITORING AGENCY NAME(S) AND ADDRESS(ES)	**10. SPONSORING / MONITORING AGENCY REPORT NUMBER**

11. SUPPLEMENTARY NOTES

The views expressed in this thesis are those of the author and do not reflect the official policy or position of the Department of Defense or the U.S. Government.

12a. DISTRIBUTION / AVAILABILITY STATEMENT Approved for public release, distribution is unlimited.	**12b. DISTRIBUTION CODE**

13. ABSTRACT

This thesis proposes a new structured methodology for incorporating the use of enterprise architecture techniques into the DoD software acquisition process, to provide a means by which interoperability requirements can be captured, defined, and levied at the appropriate time in a system's development. It discusses the necessary components of these architectural models, how these models capture our interoperability needs, and how these interoperability needs form the basis for meaningful dialogue between the DoD's acquisition and planning communities. While this methodology is applicable to many domains and functional areas, for the purposes of this thesis, the focus will be solely on software systems (including systems with embedded software) within the DoD.

14. SUBJECT TERMS Software Architectures, Enterprise Architectures, Software Engineering, Interoperability, System Integration, Software Integration	**15. NUMBER OF PAGES** 101
	16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT Unclassified	**18. SECURITY CLASSIFICATION OF THIS PAGE** Unclassified	**19. SECURITY CLASSIFICATION OF ABSTRACT** Unclassified	**20. LIMITATION OF ABSTRACT** UL

THIS PAGE INTENTIONALLY LEFT BLANK

ENGINEERING SOFTWARE FOR INTEROPERABILITY THROUGH USE OF ENTERPRISE ARCHITECTURE TECHNIQUES

Jennifer L. Parenti
Captain, United States Air Force
B.S., United States Air Force Academy, 1995

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN SOFTWARE ENGINEERING

from the

NAVAL POSTGRADUATE SCHOOL
March 2003

Author:
Jennifer L. Parenti

Approved by:
Valdis Berzins, Thesis Advisor

Richard Riehle, Second Reader

Valdis Berzins, Chairman
Software Engineering Curriculum

Chris Eagle, Chairman
Department of Computer Science

THIS PAGE INTENTIONALLY LEFT BLANK

ABSTRACT

There are many efforts underway focused on resolving the system and software interoperability problems within the Department of Defense. While several of these efforts are attempting to attack this issue using new technologies and standardization, experience suggests most of these interoperability problems are caused by deficiencies in the way we define and capture interoperability requirements within our acquisition processes and policies. In order to affect real progress towards department-wide interoperability, it will be necessary to change the methods by which interoperability is considered in the acquisition process.

Many acquisition agents within the DoD suffer from the misconception that technology alone can solve their interoperability problems. The reality is that there are many challenges within the requirements and planning processes that first must be overcome before technology can be effectively applied. Since interoperability requirements are dynamic, and often poorly understood before systems are put to use in the field, the requirements and acquisition communities must have a flexible and powerful method to communicate in order to overcome these challenges. This thesis provides a solution with which the DoD can address these fundamental gaps in our acquisition processes, thus creating an environment more conducive to software interoperability within our system of systems.

This thesis will propose a new structured methodology for incorporating the use of enterprise architecture techniques into the DoD software acquisition process, to provide a means by which interoperability requirements can be captured, defined, and levied at the appropriate time in a system's development. It will discuss the necessary components of these architectural models, how these models capture our interoperability needs, and how these interoperability needs form the basis for meaningful dialogue between the DoD's acquisition and planning communities. While this methodology is applicable to many domains and functional areas, for the purposes of this thesis, the focus will be solely on software systems (including systems with embedded software) within the DoD.

THIS PAGE INTENTIONALLY LEFT BLANK

TABLE OF CONTENTS

LIST OF TABLES AND FIGURES

LIST OF ABBREVIATIONS, SYMBOLS, AND ACRONYMS

AOC	Air Operations Center
AP	Application Protocol
ASD	Assistant Secretary of Defense
C2	Command and Control
C3I	Command, Control, Communications, and Intelligence
C4	Command, Control, Communications and Computers
C4I	Command, Control, Communications, Computers and Intelligence
C4ISP	C4I Support Plan
C4ISR	Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance
CADM	Core Architecture Data Model
CCA	Clinger-Cohen Act
CINC	Commander in Chief
CJCSI	Chairman of the Joint Chiefs of Staff Instruction
COP	Common Operational Picture
COTS	Commercial-Off-the-Shelf
CRD	Capstone Requirements Document
DoD	Department of Defense
DoDD	Department of Defense Directive
Exch	Exchange
Func	Function
GCCS	Global Command and Control System
GIG	Global Information Grid
GISRA	Government Information Securities Reform Act
GSORTS	Global Status of Resources and Training System
HQ	Headquarters
I3	Integrated Intelligence and Imagery
IER	Information Exchange Requirement
INCOSE	International Council on Systems Engineering
Info	Information
ISO	International Standards Organization
IT	Information Technology
J2	Director of Intelligence
J3	Director of Operations
J6	Director of Command, Control, Communications, and Computers
JFPO	Joint Forces Program Office
JOA	Joint Operational Architecture
JOPES	Joint Operational Planning and Execution Segments
JSF	Joint Strike Fighter

JV	Joint Vision
MNS	Mission Need Statement
ORD	Operational Requirements Document
OV	Operational View
POM	Program Objective Memorandum
Reqt	Requirement
SIEC	System Information Exchange Capability
SIER	System Information Exchange Requirement
SIPRNET	Secure Internet Protocol Routing Network
SPAWAR	Space and Naval Warfare Systems Command
SV	Systems View
TBMCS	Theater Battle Management Core Systems
TRANSCOP	U.S Transportation Command Common Operational Picture
U.S.	United States
UJTL	Unified Joint Task List
UML	Unified Modeling Language
USPACOM	United States Pacific Command

I. INTRODUCTION

Interoperability is a state, a condition in which two systems have the ability to exchange the information its users need in a meaningful manner. Achieving a state of interoperability between two systems requires detailed planning and forethought. Knowing what information is required to be exchanged and what formats the systems will support and making the proper arrangements for an interface between these systems can be a daunting task. However, it is the maintenance of this state of interoperability, which poses the real challenge. Ensuring no uncoordinated changes are made to either the systems or the environment that may affect this state of interoperability is increasingly difficult in today's world. Add several systems, systems of systems, or families of systems into this equation and the complexity grows exponentially.

To make two systems interoperable requires several detailed planning steps and multiple layers of technical understanding. For example, one needs to know the pieces of information to be exchanged, how the two systems will be employed and employed together, the physical support necessary to achieve interoperability, the components and capabilities of the two systems, and the underlying technologies on which those systems rely. To keep two systems interoperable requires a methodology through which changes in the two systems are tracked over time, to ensure a constant state of interoperability.

Several communities of distinct people and needs bring a system to fruition. Users and developers are both stakeholders in a system's development, but both with very different perspectives and attitudes towards the system. Furthermore, the people developing a system's requirements and specifications, or the people maintaining a system, may be neither the user, nor the developer. Capturing the perspectives and needs of all these communities can be difficult, and when trying to meet the needs of multiple systems, each catering to their own sets of multiple communities, the problem of interoperability can overwhelm traditional system and software development models.

In recent years, the concepts of using architectures to solve system interoperability problems have received much attention. Terms like software architecture, enterprise architecture, operational and system architecture have flooded the engineering community with hopes of tackling interoperability troubles. Department of Defense Directive 5000.1 states,

> Interoperability within and among United States forces and U.S. coalition partners is a key goal that must be addressed satisfactorily for all Defense systems to that the Department of Defense has the ability to conduct joint and combined operations successfully… The Department of Defense must have a framework for assessing the interrelationships among and interactions between U.S., Allied, and coalition systems. Mission area focused, integrated architectures shall be used to characterize these interrelationships. This end-to-end approach focuses on mission outcomes and provides further understanding of the full range of interoperability issues attendant to decisions regarding a single program or system. [15]

In my capacity as a Command and Control Interoperability Project Officer for the Joint Forces Program Office (JFPO), Space and Naval Warfare Systems Command (SPAWAR), San Diego, CA, I had the opportunity to work hands-on with several of the major interoperability and architecture initiatives that are ongoing throughout the Department of Defense. Through this work, I have concluded that the greatest benefit architecture can achieve towards interoperability is in the capturing and maintaining of system interoperability requirements. Furthermore, while many of the current initiatives have good intentions, they will fail to achieve interoperability because they do not recognize two key aspects of the relationship between interoperability and architecture: 1) there are many communities, each with distinct needs, which must work collectively to create interoperable families of systems; and, 2) a single architecture will never meet the needs of all those stakeholders.

This thesis proposes a methodology for integrating the concepts of enterprise architecture into the system and software development cycles, simplifying the currently used models into only those objects necessary for achieving the goal of interoperability. It proposes a unified repository of architectural data, but with the ability to be viewed in several forms (i.e. with the ability to create multiple architectural views), each tailorable to the needs of the different stakeholders. The power of this methodology is it provides a mechanism by which functional and interoperability requirements are captured, defined, and levied on systems based on how they will be employed. This is a dynamic process, which can accept changes to requirements, system environments, and domains; and facilitates the concepts of time-phasing, spiral development, requirements vs. capabilities, and operational vs. system needs.

A. RESEARCH QUESTIONS

This thesis will answer the following research questions:

1) Can the use of enterprise architecture throughout the software acquisition lifecycle improve the process of defining interoperability requirements for software systems?
2) How do architectural models allow a software developer to capture the evolving interoperability needs of all a system's stakeholders?
3) What architectural components are required to support the development and maintenance of interoperability requirements?
4) How does architecture modeling allow multiple software developers to synchronize the development of several independent software systems so interoperability is continuously achieved?

B. SCOPE AND LIMITATIONS

This thesis proposes a methodology for using enterprise architecture techniques to capture, define, and levy software interoperability requirements. The author recognizes that there are a multitude of other applications of enterprise architectures, even within the Department of Defense, that would require different data models, different implementation techniques, and different stakeholders. This thesis is limited only to those architectural requirements necessary for the purposes of capturing, defining, and levying software and system interoperability requirements; and, therefore omits--by design--implementation considerations like technical architecture views and technology standards.

C. ORGANIZATION

This thesis is comprised of six main chapters:

- Chapter I: Introduction

- Chapter II: Background/Previous Studies – an outline of some of the many enterprise-architecture related efforts ongoing in industry and the Department of Defense (DoD).

- Chapter III: Software & Interoperability within the DoD -- This chapter introduces the reader to the concepts of interoperability, how it fits into the software domain, and how these concepts are currently treated within the DoD. It is intended to establish a common framework for understanding of the rest of the thesis.

- Chapter IV: Enterprise Architecture for Software Interoperability -- This chapter discusses what enterprise architecture is and shows, through a storyboarded example, how proper application of these techniques leads to formation of interoperability requirements.

- Chapter V: Benefits of the Data-Driven Approach – an explanation of the benefits of using a relational database for capturing architectures, as opposed to the more commonly applied "picture" approach.

- Chapter VI: Closing Comments – This chapter provides thesis recommendations and conclusions.

II. BACKGROUND/PREVIOUS STUDIES

There are a number of enterprise architecture-related efforts going on throughout the Department of Defense and industry. To gain an appreciation for what this thesis represents, and how it fits into current activities, I have chosen to outline a few of those efforts on the following pages.

A. DOD ARCHITECTURE FRAMEWORK DOCUMENT [1]

No discussion of architecture within the DoD could begin properly without an explanation of the Department of Defense Architecture Framework Document, also known as the Command, Control, Communications, and Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR) Architecture Framework Document. This document outlines, in exhausting detail, the required elements of any DoD architecture effort, regardless of the customer or architectural need. It promotes a common framework for all architectural efforts, describing required data elements, required views of the data, and suggested applications for the architecture once completed.

The primary downfall of the DoD Architecture Framework is that it focuses on the views of the data, rather than the data itself. By mandating particular views, it forces the aspiring architect to focus on the architecture as a set of drawings and pictures, rather than focus on the relationships between the data elements, which is where the strength of the architecture lies. Because of the extent of the mandatory products, many organizations blindly go through the steps to meet the mandated requirements of the Framework document rather than take the time to understand what the architectures are and why they may be important to their organizations. This results in a 'fill the check box' approach to the architectures' creation and ultimately results in architectures that are of little use to the creator or any of the other stakeholders in the domain.

In this way, the document fails to recognize that different communities may wish to use the architectural data in different ways, thus taking away the power of a flexible, data-driven, object-oriented approach. For instance, the requirements community is likely to take a much different approach to architecture than a system developer or user or maintainer. The focus of requirements is typically on the interactions of the many organizations and systems within their area of interest, also known as the '**domain**' of their architecture. A system developer will likely focus on how his particular system interacts with other systems regardless of the domain; while a system maintainer may be interested in the evolution of a particular system over time. The current approach of the DoD Architecture Framework fails to capture these various needs of these separate communities, or '**stakeholders**.'

However, what the DoD Architecture Framework does, and does quite well, is establish a common vocabulary and structure for the creation of architectures in the DoD. This is an absolutely vital aspect of any significant architectural effort. In a sizable domain, especially one that consists of so many functional areas and distributed systems as the DoD, it is important to establish this common data model to facilitate integration of disparate architectural efforts. I will use much of the terminology of the framework document as a basis for the ontology of this thesis.

1. Architecture Views

So, to begin, the DoD Architecture Framework outlines 3 major categories of architectural products: Operational, System, and Technical. These categories have a strong foundation and are widely accepted and understood in the world of enterprise architecture development. These terms also are used as the foundational language of the proposed methodologies (Chapter IV) and are seen often throughout this thesis.

a. Operational Architecture Views

The framework describes the operational architecture view as "a description of the tasks, and activities, operational elements, and information flows required to accomplish or support a military operation." In short, the operational architecture views equates to business modeling for the DoD.

Operational architecture views are generally independent of technology, systems, or organization/force structures. In theory, they should describe how missions and functional areas (i.e. Theater Air and Missile Defense, Close Air Support, or Anti-Submarine Warfare) are accomplished from an activity and information flow standpoint, regardless of what organizations or systems are available to accomplish the mission. However, in practice, because of the products mandated by the DoD Architecture Framework, these views are often modeled around existing force structures and systems, rather than considering how the mission/functional area *should* be accomplished regardless of a particular system or force implementation.

This practice greatly reduces the reusability of the created architectures, as the information captured is often too specific to be applied to similar domains. Operational architectures ideally should be so generic that, for example, an architecture that captures the Close Air Support mission in one theater should be able to be re-used in another theater, as doctrinally the *missions* should be the same regardless of the *units* or *systems* which are implementing them.

Within the DoD Architecture Framework, operational views are described using these basic architectural elements: nodes, activities, and information exchange requirements. *Nodes* are virtual entities that represent a collection of activities (and, within the Architecture Framework, systems, as well). Nodes are places where activity occurs. Example nodes might be: Command Post, Destroyer, or Fighter Wing. The operational views in the DoD Architecture Framework tend to be node-centric, i.e. they start with development of the nodes and describe activities and interactions at the node level. After the development of the nodes comes assignment of activities to the nodes. In most DoD architectures, activities are derived from the Unified Joint Task List (UJTL)—a living list of the common activities required to perform daily and wartime missions. Requirements for nodes to exchange information are documented *Information Exchange Requirements* (IER). Typically, IERs are defined as using an information element (the description of the information being exchanged) and two nodes. While it is preferable that IERs also list the activities within the nodes that generate the need for the IER, it is often seen that activity modeling is not complete enough to meet this requirement. This focus of the operational view on the nodes and not the activities is another shortcoming of the DoD Architecture Framework.

b. System Architecture Views

The system view is "a description of systems and interconnections providing for, or supporting, warfighting functions." In short, the system architecture views are wiring diagrams, showing systems and how they interconnect.

Within the C4ISR Framework, architectural system views tend to take on the form of static representations of a given architectural domain. That is, they show specific instantiations of systems and how they are physically connected. This approach to systems architecture is, in this author's opinion, one of the major shortcomings of the C4ISR Architecture Framework Document. This approach fails to capture the correct information to achieve system interoperability.

These types of views, which show workstations and servers and circuits and routers are generally only of use at the micro-level—that is, they are useful to the personnel responsible for the maintenance of those systems locally, but generally not of much use to the actual engineering process for generating requirements and implementing them. Rather, system-to-system interoperability requires a macro view of the world. To achieve this, system architecture views must also take on a more generic approach of understanding how *system types*, as opposed to instantiations, are required to interoperate. For example, to someone responsible for documenting requirements for the Global Command and Control System (GCCS), it is less important to him to know that a GCCS system at Hickam Air Force Base (AFB) is located in Bldg A and connects to the SIPRNET via Router B, C, & D, as it is to know that GCCS needs SIPRNET connectivity to exchange information with the Theater Battle Management Core System (TBMCS.) This is one example of micro- versus macro-level system architectures.

Another shortcoming of the systems views in the DoD Architecture Framework is that it does not allow for the documentation of system capabilities versus requirements. While system requirements are an extremely important aspect of the acquisition and development processes, it does little good to the warfighter to know that two systems are *required* to exchange certain elements of information. Rather, it is more important in this case to understand whether the systems are *capable* of exchanging these information elements. This is an extremely important distinction, and a key shortcoming of the Architecture Framework that must be overcome.

System views are typically defined using *nodes*, *systems*, and *interfaces*. The concept of nodes is equivalent to that in the operational view—a collection of systems capable of performing certain functions. *Systems* are defined at the level of the architect; however, system hierarchies are not easily supported, so it is up to the architect to maintain consistency and determine at what level 'systems of systems' are to be architected. *Interfaces* represent physical connections between systems. Usually, interfaces will only be represented if there is some meaning to the interface (i.e. information can actually be exchanged over the connection.) However, this points out another shortcoming of the Architecture Framework. That is, there are few dependencies between the operational and system views; and, therefore, they can be created independent of each other, with little to no coordination between the architects. Therefore, just because two systems have IERs in the operational view and interfaces in the systems view, this does not necessarily mean that those IERs are supported by the interfaces—the two systems may not be interoperable. This is another issue addressed by the methodology proposed in this thesis.

c. Technical and "All" Architecture Views

The technical architecture view is "the minimal set of rules governing the arrangement, interaction, and interdependence of system parts or elements, whose purpose is to ensure that a conformant system satisfies a specified set of requirements." That is, the technical architecture is the standards on which the systems within the architecture are based. Often, as in the case of the DoD, technical architectures are used to levy technology standards onto the systems that fall under its purview. Technical architectures focus on implementation decisions of standards across the enterprise domain. As this thesis focuses on the requirements definition process, it will not discuss technical architectures.

The "All" architecture views are two additional products also mandated by the framework. They are primarily administrative in nature and serve to provide introductory and summary information of the architecture as well as provide an architectural dictionary of terms and acronyms. These views are useful in giving an overall description of the architecture and in attempting to gain data commonality; but disparate sets of dictionaries, which require no coordination between them, will never achieve the goal of an integrated data dictionary. This is another reason why a data-centric approach to architectures, using an object-oriented common architectural repository (database) is so necessary.

2. C4ISR Core Architecture Data Model [9]

The C4ISR Core Architecture Data Model (CADM) is the companion document to the C4ISR Architecture Framework. It describes the basic set of standardized entities that should be used when building C4ISR architectures. In short, the CADM describes every object, every attribute, every relationship contained within the C4ISR Architecture Framework.

The CADM is designed to provide a common approach for organizing and portraying the structure of architecture information. By facilitating the exchange, integration, and comparison of architecture information throughout DoD, this common approach should help improve C4ISR architecture interoperability and reusability.

It is in the interest of supportability and tractability of enterprise architectures that a common set of architectural data elements be developed—whether as part of the DoD mandate, or in any commercial architectural endeavor as well. This provides a basis for information sharing between architectures, with an end goal of someday being able to integrate existing architectures into a common database. It is important, for this reason that any DoD architecture efforts not deviate from the CADM. Appendix B shows the relationship between the proposed architecture data model and the CADM, and demonstrates there are several shortcomings to the CADM that could be easily incorporated into the current model to achieve better data fidelity and facilitate a more dynamic incorporation of interoperability requirements and capabilities.

B. JOINT OPERATIONAL ARCHITECTURE

The Joint Operational Architecture (JOA) is the DoD's premiere attempt at creating a common operational architecture. The JOA writers took a similar approach to the author in that they first defined their domain (all DoD) and functional areas (using Joint Vision 2010 goals). These concepts will be discussed further in Chapter IV. The JOA efforts are the best observed thus far in the DoD in attempts to create top-down operational architectures (requirements must come from the top); unfortunately, it will fall short of being useful for defining interoperability requirements due to lack of proper resourcing and support and because they have failed to take a data-driven approach to their architecture. Their static 'picture' approach is not dynamic enough to be of benefit to the acquisition community. The proposed methodology will address an alternative to these efforts.

C. GLOBAL INFORMATION GRID (GIG) ARCHITECTURE

The GIG Architecture effort, led by the Assistant Secretary of Defense for Command, Control, Communications, and Intelligence (ASD(C3I)) has quickly become the leading architectural effort in the Department of Defense. The GIG effort is currently attempting to do what few other architectural efforts have attempted in the past: it is attempting to create a domain-wide architecture by integrating existing architectural products.

This undertaking is proving extremely difficult as the architects are finding that the existing architectural products suffer from the weaknesses of the DoD Architecture Framework discussed earlier (i.e. they lack common language, structure, data formats, etc.) However, the GIG Architecture effort has two strengths/advantages over other architectural efforts: 1) it is well resourced; 2) it has the ability to take a domain-wide approach to the architecture as it comes from the highest levels of the Defense Department. This top-level view will prove extremely helpful in the creation of future architectures that will certainly be based on the efforts of the GIG architects.

D. AP-233

Application Protocol 233 (AP-233) is the more common name for ISO 10303-233, an emerging ISO systems engineering data exchange standard. It is relevant to this thesis work in that it also will attempt to allay system interoperability problems that are created when simultaneous development efforts are not coordinated. In the systems engineering community, there exist many tools designed to aid systems engineers in the capturing of requirements, capabilities, and physical implementations. In this sense, these tools are not unlike the many tools that exist to capture enterprise architectures. Over the years, these tools have been developed for specific projects at specific times with little thought or attention given to the idea that eventually, it might become important to exchange information between them.

Now, the systems engineering community, namely through the International Council on Systems Engineering (INCOSE) and ISO, is attempting to bring interoperability to these tools through the creation of a common data standard. This work is comparative to the creation of the common architectural data repository proposed in this thesis and the two share many similar aspects, primarily in scope and purpose.

AP-233 Working Draft 5 consists of several domains: requirements; functional design; physical design; graphical representation and layout; traceability management; configuration management; and industry process [13]. This design mirrors the intent of the proposed common architectural data repository (which will be discussed more in Chapters IV and V) especially in that it differentiates between requirements, capabilities (functional design), and implementation (physical design). But also in that it provides common standards for graphical representation (a key aspect of any architectural effort—including the proposed methodology), traceability and configuration management (the reasons for proposing a centralized data model) and also in that it recognizes the many needs of its stakeholders through its ‘industry process’ domain. This domain focuses on risk management and other user-centric issues.

AP-233 is still currently a draft standard and is in coordination. Recent publications indicate it may be published sometime in 2003. However, Appendix C contains that latest information on AP-233 and how the proposed architectural data model relates.

THIS PAGE INTENTIONALLY LEFT BLANK

III. SOFTWARE & INTEROPERABILITY WITHIN THE DOD

Software and interoperability are inextricably linked—especially in today's world of high technology and software-driven communications systems. But the complexities behind what it means to be interoperable, how we define interoperability requirements, and how they get levied onto our software-intensive systems are not always appreciated.

A. DEFINITION OF INTEROPERABILITY

According to CJCSI 6121.01b, Interoperability and Supportability of National Security Systems, and Information Technology Systems, *interoperability* is defined as: (1) The ability of systems, units, or forces to provide services to and accept services from other systems, units, or forces and to use the services so exchanged to enable them to operate effectively together, and (2) The condition achieved among communications-electronics systems or items of communications-electronics equipment when information or services can be exchanged directly and satisfactorily between them or their users. In short, interoperability is achieved when every user has the ability to get the services or information they require in any situation and are able to use that information in the successful completion of their mission.

This definition of interoperability is predicated upon the existence of an understanding of the requirements to be interoperable. In fact, CJCSI 6212.01b goes further on to state that for the purposes of this instruction, the degree of interoperability will be determined by the accomplishment of the proposed Information Exchange Requirements. In that sense, it would be impossible to understand to what extent units, systems, or users are interoperable without knowledge of their requirements to be so. This highlights the importance of a methodology to capture interoperability requirements and levy these requirements effectively on our systems.

B. CURRENT INTEROPERABILITY SITUATION IN THE DOD

Interoperability of DoD weapons and communications systems is among the top priorities of all our Unified CINCs. It is a problem that continues to grow, and our reliance on information and information superiority in modern warfare will ensure its importance as we move deeper into the 21st century. Concepts such as Joint Vision 2010 and Joint Vision 2020 already rely heavily on the pre-existence of joint interoperable systems. These concerns are exacerbated as we consider a more global schema of federal and coalition (i.e. international) interoperability.

But system interoperability is not just an exercise for the system or software developer. Many times, interoperability problems cannot be overcome by technology alone. According to Hamilton and Murtagh [3], “compatible systems, doctrine, and policy must exist”. And, a common, data-driven architectural approach is a flexible, maintainable methodology to bring these three very different, but very necessary and related aspects of the system development process together. Hamilton and Murtagh go further to state that requirements engineering is the first step towards achieving system interoperability. [3]

C. HOW THE DOD DEFINES INTEROPERABILITY REQUIREMENTS

CJCSI 3170.01b defines three primary documents it uses to capture requirements: the Mission Need Statement (MNS); Capstone Requirements Document (CRD); and Operational Requirements Document (ORD). Warfighter mission needs are defined in broad operational terms in a MNS document. Subsequently, the needs expressed in the MNS are developed into requirements in the forms of CRDs and ORDs. CRDs act to provide ORD development guidance for a mission area that forms a system of systems or family of systems. ORDs translate the MNS and CRD requirements into detailed, refined performance capabilities for a specific proposed system. [10]

Currently, an interoperability requirement is captured as an element of the CRD and ORD architectures known as an Information Exchange Requirement (IER). IERs are defined as part of the Operational View of the architecture and are specifically captured in an architectural product known as the Operational Information Exchange Matrix. Figure 1 shows the data requirements of the Information Exchange Requirement, as depicted in the DoD Architecture Framework V2.0.

The Operational Information Exchange Matrix (OV-3), shown in Figure 1, is one of the many architectural products mandated by the DoD Architecture Framework. In fact, it is a mandatory part of all CRDs, ORDS, and C4I Support Plans (C4ISP).[1] These actions are a good first step towards capturing system and software interoperability requirements, but they have to date proven insufficient.

INFORMATION DESCRIPTION					INFORMATION SOURCE		INFORMATION DESTINATION		INFORMATION EXCHANGE ATTRIBUTES		
OPERATIONAL INFORMATION ELEMENT	DESCRIPTION	MEDIA	SIZE	UNITS	OPERATIONAL ELEMENT & ACTIVITY		OPERATIONAL ELEMENT & ACTIVITY		FREQUENCY, TIMELINESS, THROUGHPUT	SECURITY	INTEROPERABILITY REQUIREMENTS
NAME/IDENTIFIER	DEFINITION	DIGITAL, VOICE, TEXT, IMAGE, ETC.	RANGE LIMITS	FEET, LITERS, INCHES, ETC.	IDENTIFIER OF PRODUCING OE	PRODUCING ACTIVITY	IDENTIFIER OF CONSUMING OE	CONSUMING ACTIVITY			

Figure 1. Operational Information Exchange Matrix (OV-3) – Representative Format

D. MAJOR SHORTCOMINGS OF CURRENT DOD PROCESSES

There are many shortcomings in both the DoD system acquisition process and the current handling of architectures that are preventing meaningful usage of the IERs identified to date. First, despite the mandatory architectural products, many systems are being fielded without CRDs, ORDs and C4ISPs. This indicates a lack of discipline in our system acquisition processes in that we continue to field systems that do not have validated requirements. Additionally, while the system *developers* are asked to create the C4ISPs, they are not

[1] C4ISPs contain all the information required to sustain a system (logistics plans, training plans, architectural plans) throughout its lifecycle. They are required at certain milestone decision steps for each DoD system, in accordance with the Government Information Systems Reform Act (GISRA) and the Clinger-Cohen Act (CCA).

responsible for creating the operational architecture products (*requirements*-driven) that are contained within them. These products are the responsibility of the *requirements* community.

Second, the architectural information that is captured in these documents is located within the confines of a paper document. There is no meaningful way to integrate this information with that contained within similar documents. These documents meet a periodic review cycle, but all too often they become 'shelfware' never to be referenced again after their initial creation.

Third, there exists no central repository for the architectural information that does exist within the CRDs, ORDs, and C4ISPs; and, therefore, there exists no methodology for ensuring consistency between them. For instance, there is no formal method for ensuring that an IER that the TBMCS ORD has documented with GCCS is, in turn, documented in the GCCS ORD in reverse. (In truth, there exists to date several IERs between TBMCS and GCCS that cannot be documented in reverse, as there exists no GCCS ORD) And, because these architectural products cannot be maintained as 'living' documents with dynamic updating of requirements and consistency between systems and functional areas, proper allocation of requirements to systems cannot be accomplished.

Demonstrating these points, during a survey at HQ USPACOM conducted by a combined CINC Interoperability/ Joint Forces Program Office team on 1 March 2000, approximately sixteen documented or ongoing architecture efforts were revealed across the J2, J3, and J6. Each effort was separate and distinct. Each was separately funded and initiated. No centralized data repository existed even between these components of a single unified command headquarters. [6] And since that time, the number of these disparate architecture efforts has grown exponentially as architectures have become a mandatory part of the system acquisition process.

E. SOFTWARE ENGINEERING AND INTEROPERABILITY

Interoperability is accomplished by first identifying data needed by other users or systems, and then by arranging to share that data quickly enough that it is still useful upon receipt by those other users or systems. [2] Data, data exchanges, interfaces and data-driven applications are all fall within the responsibility of the modern software engineer. Data formats and database design, system interfaces designs, application design and implementation—all of these activities fall into the realm of software engineering.

Some might argue that many interoperability problems are hardware problems. But, diverse hardware-based communications systems require an overall software architecture in order to interoperate. [3] Added to this, modern communications systems (which bear the brunt of data requests and interchange problems) are software-intensive. Hardware is not easily changed, and fielded hardware systems often cannot be wholly replaced. Therefore, as a practical matter, interoperability is more easily achieved through software. [3]

And, so, as today's systems become more complex and more inter-related, and the requirements for seamless information flows and transfers grow faster than the technology, it is clear that software engineers (and all system engineers) need a better architecture-based system to capture and define interoperability requirements.

THIS PAGE LEFT INTENTIONALLY BLANK

IV. ENTERPRISE ARCHITECTURE FOR SOFTWARE INTEROPERABILITY

An object-oriented approach to enterprise architectures may be the solution that can bring system developers, requirements experts, policy, and doctrine together to form a dynamic approach to the systems and software requirements engineering processes and allow these all-too-disparate communities to find a common ground for communication.

A. DEFINITION OF ENTERPRISE ARCHITECTURE

In its most basic form, an architecture is simply a description of objects and the relationships between these objects. Any system, software, enterprise, or other architecture can be described so.

Enterprise architecture provides a top-level model of how information flows across the organizations within the enterprise domain. It identifies the key nodes, potential constraints, and the relationships between these nodes. It is a cornerstone to integrating or updating technologies and understanding what data is needed where and when. [6] In short, enterprise architecture equates to a business modeling method.

As with many methods, enterprise architectures can be used to demonstrate different ideas and concepts depending on who is using them, and how they are used. On one side, they can be used to describe business processes, information flows, and activities. In this sense, enterprise architecture provides the underlying framework, which defines and describes the platform required by the enterprise to attain its objectives and achieve its vision. [4] In this way, enterprise architectures can be used to capture a common perspective--a common vision--of how a business domain should function. The objects of the architecture may be activities, grouped together into roles or functions, with required information flows representing the relationships between the objects.

From another angle, enterprise architectures can be used to describe information technology (IT) capabilities, their networks, and their functions. In this case, the architecture provides a networking diagram, which defines the capabilities the enterprise has to achieve its objectives and vision. In this way, enterprise architectures can be used to capture existing capabilities and future needs in any networking domain. The objects of this architecture may be systems, their subcomponents, and the transactions that are required and/or supported between these components.

B. BENEFITS OF ENTERPRISE ARCHITECTURE

Enterprise architectures provide a framework for modeling of business practices and allocating systems to that framework. The techniques are extremely flexible and can be designed to benefit a number of different communities, even within the same business domain. For instance, enterprise architecture techniques can be used to capture warfighting doctrine from the planning and requirements communities just as easily as it can be used to demonstrate system-to-system interactions. And, it is just as easily adaptable to software integration (relationship between software components and modules) as it is to system interoperability. This is realized through recognition that the architecture is not a collection of paper drawings, but is rather a structured database, by which the data elements can be related and viewed in any number of ways, depending on the interests of each particular user (or sets of users).

When combined with the use of an object-oriented, relational architecture database that can be easily updated, maintained, and reused, there are many benefits that can be realized over the current processes. First, repeated duplication of efforts and multiple data requests would be reduced. Instead of multiple architectural efforts which are geared towards a specific customer, by incorporating a data-centric, central repository approach, all architectural efforts eventually contribute to the corporate knowledge of the entire community. And, by embedding the data and the use of that data into the business processes of the organization, the demand for (static) products is reduced, if not eliminated. [6] Furthermore, enterprise architecture planning considers both the strategic and tactical need for information exchange in supporting the organization's mission. Using a data-centric approach, time attributes would provide the necessary information to improve contingency and resource planning and allocations. [5]

C. NEW ARCHITECTURAL MODEL

The proposed architectural model uses an enterprise architecture approach to define, capture, levy, and maintain system interoperability requirements. It is a data-centric, object-oriented architectural model that focuses on the relationships between architectural elements, not picture representations. Furthermore, it simplifies the current DoD models to capture only those pieces of information required for achieving interoperability.

The model is designed to recognize the distinct needs of all a system's stakeholders, allowing for different architectural foci, constructed from the same underlying data. It is also tailored to meet the ultimate goal of interoperable systems and forced, structured coordination between the planning/requirements and acquisition communities.

Finally, the model spans time, allowing the various communities to incorporate the concepts of time-phased requirements and spiral development.

D. ELEMENTS OF PROPOSED ARCHITECTURAL MODEL

1. Step One: Establish the Domain and All Its Stakeholders

In order to create a valid architecture, it is vital to have a clear understanding of the environment that the architecture is intended to model and the questions/issues the architecture is intended to answer. Example domains could be: a hospital; Air Force Command and Control; a collection of integrated software components, like the Common Operating Environment, or Microsoft Office; Theater Air and Missile Defense; the entire Department of Defense; a single software application; or, all Federal Government Agencies. Domains can be very large or very small, depending on the interest of the architect. There is no right or wrong answer or approach, as long as the intentions are clear from the beginning and consistency is maintained throughout. The domain also creates limits and brings discipline to the architectural process.

The desired result of this step is a definition of the domain of the architecture and a list of all the stakeholders and their responsibilities with respect to the environment being modeled.

For this thesis, I have chosen to storyboard the domain of home security. Within that domain the following groups own some kind of stake in the systems of that domain:

Stakeholder	Responsibilities
Homeowner	Generates requirements based on his security desires
Monitoring Service	Provides service based on homeowner demands
Emergency Response Services	Responds as needed, maintains public systems
System Developers	Responsible for engineering system components
System Installers	Responsible for installing system components
System Maintainers	Responsible for maintaining system components

Table 1. Architectural Domain Stakeholders

a. Users

The homeowner, monitoring service and emergency response services each represent potential users of the developed system, and, thus, ultimately will drive the requirements of the domain. In large commercial domains, such as this, it is nearly impossible to reach a consensus of requirements between these disparate groups. Additionally, even with consensus, it may not be feasible for the developer to include all users' requirements in a single release. In the Defense domain, where the user base is much smaller and more easily accessible than usually found in commercial industry, and the systems and applications more tailored to specific functions within the domain, it is much more likely to see a user directly involved in the requirements generation aspect of a system—in fact, it is a basic tenet of Defense System Acquisition. But, even in this semi-controlled environment, management of ever-changing priorities and disparate user communities tracking and monitoring of requirements is an enterprise-wide challenge.

In the architectural process, it is the users'/requirements community's responsibility to capture the requirements of their domain as elements of the architecture. The proposed architectural model aids the requirements community by allowing them to capture their domain requirements (activities, information exchanges, etc.) in a central architectural repository, which can then be shared with other communities of interest. Information exchange requirements, which by definition occur between two or more activities, are automatically assigned to the appropriate each activity (a required data element) eliminating the mismatches between multiple paper requirements documents. Combined with a common data dictionary, this approach also will prevent confusion and miscommunication between disparate requirements and user communities.

b. Developers

System developers are responsible for implementing requirements, as defined by the users or designated requirements community, into capabilities. In today's defense and commercial environments, it is often the case that the requirements of a domain will be met by multiple systems (and, thus, system developers) and that often a single system will meet partial requirements of many domains. Therefore, the system developer needs to understand how his system fits into the integration of multiple systems within a single domain, and how it fits into the integration across multiple domains.

With respect to the architecture, it is the responsibility of the system developer to track the requirements that have been levied on his system and their implementation. He reads the requirements data, as defined by the user community, and submits to the central architecture repository his plans to implement these requirements as capabilities. In this way, users and other interested communities can track when capabilities (including interoperabilities between multiple systems) will be available to them.

c. System Installers and Maintainers

A centralized system architecture is even of use to the individual system installers and maintainers, as it provides them cohesive insight into the current system interactions and how those may change in the future. System installers and maintainers gain insight from the architecture through a documented understanding of how the system is intended to be employed and the other systems with which it is supposed to interact. Future and planned capabilities and requirements identify potential needs for installation and maintenance training impacts.

2. Step Two: Understand Your Business

Within each domain, there may be several functional areas or mission areas that must be further defined. These functional areas can be architected independently, but will normally be linked via common requirements and common system implementations. Within each functional area, it is necessary to accomplish the following:

- List/define all the activities required to execute the processes within the functional area (hierarchically group activities, if required)
- Determine the necessary information exchanges between those activities
- Smartly aggregate activities into roles/nodes

The desired result of these steps is a complete activity model for each functional area within the domain, grouped into actors/roles/nodes, with information exchanges identified between these groupings.

a. Define Functional Areas

The identification of functional areas within the architectural domain is an optional step, but particularly useful for any larger scale architectural efforts. Functional areas provide a decomposition of the domain into smaller-scale and, thus, more manageable architectural projects, allowing for better organization of and control over the architectural process, as a whole.

If one was architecting the Department of Defense, example functional areas could be Theater Air and Missile Defense, Command and Control, or Close Air Support. If one was architecting a Microsoft Office competitor, example functional areas could be word processing, graphics, spreadsheet design, or messaging.

It does not matter how the architectural effort is decomposed, as long as the breakout is applied consistently throughout the architecture. The identification of the number and scope of the functional areas belongs in the realm of the requirements community, but should be agreed to by all stakeholders. For smaller-scale efforts, the decomposition may not be necessary, if proper configuration management of the architectural elements can be maintained using a more global construct. However, all the activity modeling still applies.

In the continuing example of Home Security, I have chosen to architect the following three functional areas: Intruder Detection & Response; Fire Detection & Response; and Flood Detection & Response.

Functional Area	Description
Intruder Detection & Response	Home/Business Security. To monitor and detect unauthorized entry into the secured area and sound alarms/alert authorities, as necessary.
Fire Detection & Response	General Security. To detect smoke and/or fire within the monitored area and sound alarms/alert authorities, as necessary.
Flood Detection & Response	Home/Business Security. To detect flood conditions (i.e. excess water levels) within the monitored area and alert authorities, as necessary.

Table 2. Functional Area Descriptions

b. Define Activities Required to Execute the Functional Area

The identification of the activities required to execute each functional area is the most critical aspect of the architecture development. It is the area that will require the most research and most thorough understanding of the domain. It is also the most likely area of contention and need for group consensus, and often the most time-consuming. The activities will serve eventually as the fundamental basis for all other architectural products, and, therefore, must be carefully considered and constantly reviewed to ensure they accurately portray the functional area and the architect's desired product.

As with other architectural elements, there is no right or wrong way to define an activity, as long as the standard is applied consistently to the entire architecture. These activities eventually will serve as endpoints for interoperability requirements and as the basis for system requirements and capabilities. Their use towards these ends must also be considered in the architecture's development to ensure the right level of detail is captured in the activity model.

Risks include creating an activity model that is too high-level and cannot support requirements definition, as there is not enough detail to assign the requirements to any one particular node or system. Furthermore, too much detail can slow the architecture process and create a scalability factor that makes the rest of the effort intractable. These risks can be reduced by taking a hierarchical approach to the activity modeling which allows for the data to be viewed at whichever level of detail is appropriate to the user. For instance, if architecting the functional area of Global Command and Control, one might look at high-level activities such as Deployment Planning, Situational Awareness, and Intelligence Gathering and the interactions between these activities. But, one may want to dive deeper to find out that within deployment planning are subactivities, such as personnel deployment processing, equipment transport, and in-theater resupply. The Universal Joint Task List (UJTL), the DoD's listing of all warfighting tasks, is an excellent example of an existing, hierarchically grouped activity model.

In the area of Home Security, the following activities were identified for each of the functional areas:

Intruder Detection	Fire Detection	Flood Detection
Detect Door Opening	Detect Smoke	Detect Flood
Detect Window Opening	Detect Heat	Notify Monitoring Service
Activate Alarm	Activate Alarm	Notify Homeowner
Notify Monitoring Service	Notify Monitoring Service	
Notify Homeowner	Notify Fire Department	
Notify Police	Notify Homeowner	
Arm System	Respond to Fire/Alarm	
Disarm System		
Investigate/Respond to Alarm		

Table 3. Initial Activity List

Each functional area should not draw from a separate and distinct list of activities, however. For instance, it is not necessary for each functional area to contain the activity "Notify Homeowner." When the initial activity list is created, it is necessary to design it in such a way as to promote sharing across the functional areas. Creating an activity tree that is easily navigable by all parties is one way to promote this kind of cooperation. In the following tables, I demonstrate one method for creating such a hierarchy.

Starting with the Intruder Detection & Response area, the listed activities are grouped into similar categories.

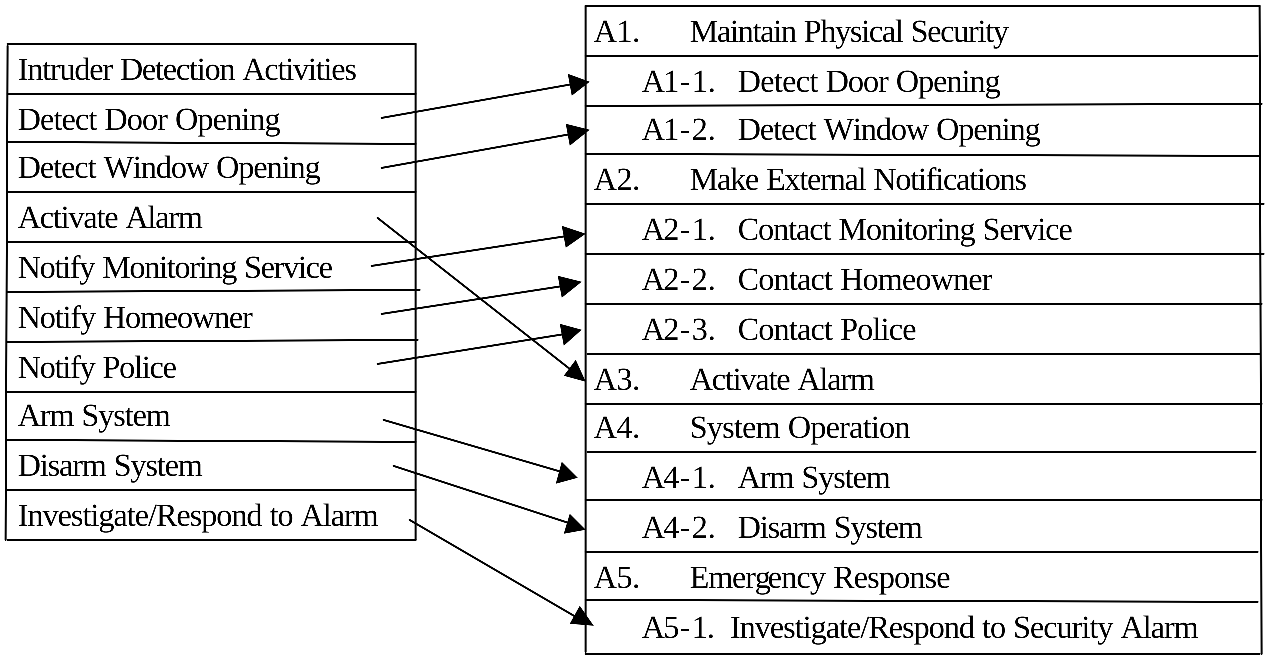

Intruder Detection Activities
Detect Door Opening
Detect Window Opening
Activate Alarm
Notify Monitoring Service
Notify Homeowner
Notify Police
Arm System
Disarm System
Investigate/Respond to Alarm
A1. Maintain Physical Security
A1-1. Detect Door Opening
A1-2. Detect Window Opening
A2. Make External Notifications
A2-1. Contact Monitoring Service
A2-2. Contact Homeowner
A2-3. Contact Police
A3. Activate Alarm
A4. System Operation
A4-1. Arm System
A4-2. Disarm System
A5. Emergency Response
A5-1. Investigate/Respond to Security Alarm

Figure 2. Intruder Detection Activity Groupings

As this is a new architecture effort, this list forms the basis for a universal activity list. Other activities to be defined for other functional areas will build off this list such that common activities need not be defined twice. If this were a modification or addition to an existing architecture, it would be important during the activity definition phase to ensure that no duplicate activities were being added to the system. This requires research and discipline on the part of the architect in understanding the existing architectural elements.

Continuing this task through the other functional areas, the following list of activities is created:

- A1. Maintain Physical Security
  - A1-1. Detect Door Opening
  - A1-2. Detect Window Opening
  - A1-3. Detect Smoke
  - A1-4. Detect Heat
  - A1-5. Detect Flood
- A2. Make External Notifications
  - A2-1. Contact Monitoring Service
  - A2-2. Contact Homeowner
  - A2-3. Contact Police
  - A2-4. Contact Fire Department
- A3. Activate Alarm
- A4. System Operation
  - A4-1. Arm System
  - A4-2. Disarm System
- A5. Emergency Response
  - A5-1. Investigate/Respond to Security Alarm
  - A5-2. Respond to Fire Alarm

Table 4. Combined Activity List

After the initial round of activity modeling is complete, it is important to vet the requirements through as many appropriate stakeholders as possible. One set of activities that will likely to have been missed—and are of utmost importance to the developer, and thus, the software engineer—is the set of *derived* activities. Derived activities are those that may not be implicitly required but become necessary to fully exercise the domain. One example of a derived activity is a decision point. For instance, in the above case, while the security system may want to automatically notify the monitoring service in the case of any anomaly, there may be some user intervention required to make a decision as to whether the situation warrants homeowner or emergency services to respond. It is important to capture this decision-making activity as it is not only part of the use-case for the functional area, but because it will generate information exchange requirements itself. The decision was made to group these decision points separately from the notification activities, and, therefore, the final list of activities follows.

A1. Maintain Physical Security
- A1-1. Detect Door Opening
- A1-2. Detect Window Opening
- A1-3. Detect Smoke
- A1-4. Detect Heat
- A1-5. Detect Flood

A2. Make External Notifications
- A2-1. Contact Monitoring Service
- A2-2. Contact Homeowner
- A2-3. Contact Police
- A2-4. Contact Fire Department

A3. Activate Alarm

A4. System Operation
- A4-1. Arm System
- A4-2. Disarm System

A5. Situation Analysis/Decision Point
- A5-1. Decide if Homeowner Intervention is Required
- A5-2. Decide if Emergency Response is Required

A6. Emergency Response
- A6-1. Investigate/Respond to Security Alarm
- A6-2. Respond to Fire Alarm

Table 5. Final Activity List

c. Define Information Exchange Requirements Between Activities

Once an activity list is in relatively final form[2] the next step to the architecture's development is to define the information exchange requirements between those activities. This task should be completed without consideration of current system capabilities or organizational structure. These will be considered in a later step. Rather, this should be constructed in an idealistic manner of how things *should* work, as opposed to how they do. In the DoD, joint

[2] Architectures are, by definition, living documents. However, in the initial development phase it is useful to benchmark certain elements in order to create a baseline of architectural elements from which to grow. This is especially important with respect to the activities, as they form the foundation for the rest of the architecture's development. While modifications to the activity list are possible, it is not recommended until the architecture is more mature (i.e. the first round of inputs is completed for the rest of the elements.)

doctrine is the key to determining interoperability requirements. Doctrine tells us how to fight and how we fight determines interoperability requirements. Policy sets the bounds on acceptable doctrine. [3]

In defining IERs, it is important to document not only the activities (endpoints of the IER,) but also the Information Element that is to be exchanged. In cases where the same set of activities is exchanging multiple pieces of information, this should be considered as multiple IERs; and, likewise, in cases where one Information Element is being exchanged between multiple sets of activities, this, too, should be considered as multiple IERs. However, in cases where the same activities are exchanging the same Information Element in different functional areas of the same architecture, this need not be captured twice, as the IER list will be available to all architects within the domain, just as the activity list.

Use cases are particularly useful in the identification of IERs, and it is the author's methodology of choice. The key is to run through all the possible scenarios in generating the IER list. But, regardless of the methods, the result of this task is a listing of all the IERs applicable to the architectural domain.

Continuing the example of home security, a typical intruder scenario was developed to identify the necessary IERs.

1) A door or window opening is detected.

2) The alarm is sounded and the monitoring service notified.

3) The monitoring service decides whether to notify the homeowner directly.

4a) If 3) is no, the alarm is reset and scenario stops.

4b) If 3) is yes, the homeowner is notified.

5) The monitoring service (with or without the input of the homeowner) decides of the authorities need to be contacted.

6a) If 5) is no, the alarm is reset and the scenario stops.

6b) If 5) is yes, the authorities are contacted.

7) The authorities respond to the alarm.

Based on this scenario, the following list of IERs was generated:

	Originating Activity	Receiving Activity	Information Element
1	A1-1. Detect Door Opening	A2-1. Contact Monitoring Service	Door Status
2	A1-1. Detect Door Opening	A3. Activate Alarm	Door Status
3	A1-2. Detect Window Opening	A2-1. Contact Monitoring Service	Window Status
4	A1-2. Detect Window Opening	A3. Activate Alarm	Window Status
5	A2-1. Contact Monitoring Service	A5-1. Homeowner Intervention Decision	Alarm Notification
6	A5-1. Homeowner Intervention Decision	A2-2. Contact Homeowner	Intervention Decision
7	A5-1. Homeowner Intervention Decision	A2-2. Contact Homeowner	Alarm Notification
8	A2-2. Contact Homeowner	A5-2. Emergency Intervention Decision	Alarm Notification
9	A5-2. Emergency Intervention Decision	A2-3. Contact Authorities	Intervention Decision
10	A5-2. Emergency Intervention Decision	A2-3. Contact Authorities	Alarm Notification
11	A2-3. Contact Authorities	A6. Emergency Response	Alarm Notification

Table 6. Intruder Detection IER List

Following in a similar fashion for the fire detection and flood detection functional areas, the following IER Lists were generated:

	Originating Activity	Receiving Activity	Information Element
1	A1-3. Detect Smoke	A2-1. Contact Monitoring Service	Alarm Notification
2	A1-3. Detect Smoke	A2-3. Contact Authorities	Alarm Notification
3	A1-3. Detect Smoke	A3. Activate Alarm	Smoke Detection
4	A1-2. Detect Heat	A2-1. Contact Monitoring Service	Alarm Notification
5	A1-2. Detect Heat	A2-3. Contact Authorities	Alarm Notification
6	A1-2. Detect Heat	A3. Activate Alarm	Heat Detection
7	A2-1. Contact Monitoring Service	A5-1. Homeowner Intervention Decision	Alarm Notification
8	A5-1. Homeowner Intervention Decision	A2-2. Contact Homeowner	Intervention Decision
9	A5-1. Homeowner Intervention Decision	A2-2. Contact Homeowner	Alarm Notification
10	A2-3. Contact Authorities	A6. Emergency Response	Alarm Notification

Table 7. Fire Detection IER List

	Originating Activity	Receiving Activity	Information Element
1	A1-5. Detect Flood	A2-1. Contact Monitoring Service	Flood Notification
2	A2-1. Contact Monitoring Service	A5-1. Homeowner Intervention Decision	Alarm Notification
3	A5-1. Homeowner Intervention Decision	A2-2. Contact Homeowner	Intervention Decision
4	A5-1. Homeowner Intervention Decision	A2-2. Contact Homeowner	Alarm Notification
5	A2-2. Contact Homeowner	A6. Emergency Response	Alarm Notification

Table 8. Flood Detection IER List

Just as with the activity lists, if these functional areas are architected independently, it may be necessary to evaluate the IERs together to eliminate duplicity. The table below shows the aggregate IER List:

	Originating Activity	Receiving Activity	Information Element
1	A1-1. Detect Door Opening	A2-1. Contact Monitoring Service	Door Status
2	A1-1. Detect Door Opening	A3. Activate Alarm	Door Status
3	A1-2. Detect Window Opening	A2-1. Contact Monitoring Service	Window Status
4	A1-2. Detect Window Opening	A3. Activate Alarm	Window Status
5	A2-1. Contact Monitoring Service	A5-1. Homeowner Intervention Decision	Alarm Notification
6	A5-1. Homeowner Intervention Decision	A2-2. Contact Homeowner	Intervention Decision
7	A5-1. Homeowner Intervention Decision	A2-2. Contact Homeowner	Alarm Notification
8	A2-2. Contact Homeowner	A5-2. Emergency Intervention Decision	Alarm Notification
9	A5-2. Emergency Intervention Decision	A2-3. Contact Authorities	Intervention Decision
10	A5-2. Emergency Intervention Decision	A2-3. Contact Authorities	Alarm Notification
11	A2-3. Contact Authorities	A6. Emergency Response	Alarm Notification
12	A1-3. Detect Smoke	A2-1. Contact Monitoring Service	Alarm Notification
13	A1-3. Detect Smoke	A2-3. Contact Authorities	Alarm Notification
14	A1-3. Detect Smoke	A3. Activate Alarm	Smoke Detection
15	A1-2. Detect Heat	A2-1. Contact Monitoring Service	Alarm Notification

16	A1-2. Detect Heat	A2-3. Contact Authorities	Alarm Notification
17	A1-2. Detect Heat	A3. Activate Alarm	Heat Detection
18	A2-1. Contact Monitoring Service	A5-1. Homeowner Intervention Decision	Alarm Notification
19	A5-1. Homeowner Intervention Decision	A2-2. Contact Homeowner	Intervention Decision
20	A5-1. Homeowner Intervention Decision	A2-2. Contact Homeowner	Alarm Notification
21	A2-3. Contact Authorities	A6. Emergency Response	Alarm Notification
22	A1-5. Detect Flood	A2-1. Contact Monitoring Service	Flood Notification
23	A2-1. Contact Monitoring Service	A5-1. Homeowner Intervention Decision	Alarm Notification
24	A5-1. Homeowner Intervention Decision	A2-2. Contact Homeowner	Intervention Decision
25	A5-1. Homeowner Intervention Decision	A2-2. Contact Homeowner	Alarm Notification
26	A2-2. Contact Homeowner	A6. Emergency Response	Alarm Notification

Table 9. Aggregate IER List

	Originating Activity	Receiving Activity	Information Element
1	A1-1. Detect Door Opening	A2-1. Contact Monitoring Service	Door Status
2	A1-1. Detect Door Opening	A3. Activate Alarm	Door Status
3	A1-2. Detect Window Opening	A2-1. Contact Monitoring Service	Window Status
4	A1-2. Detect Window Opening	A3. Activate Alarm	Window Status
5	A2-1. Contact Monitoring Service	A5-1. Homeowner Intervention Decision	Alarm Notification
6	A5-1. Homeowner Intervention Decision	A2-2. Contact Homeowner	Intervention Decision
7	A5-1. Homeowner Intervention Decision	A2-2. Contact Homeowner	Alarm Notification
8	A2-2. Contact Homeowner	A5-2. Emergency Intervention Decision	Alarm Notification
9	A5-2. Emergency Intervention Decision	A2-3. Contact Authorities	Intervention Decision
10	A5-2. Emergency Intervention Decision	A2-3. Contact Authorities	Alarm Notification
11	A2-3. Contact Authorities	A6. Emergency Response	Alarm Notification
12	A1-3. Detect Smoke	A2-1. Contact Monitoring Service	Alarm Notification
13	A1-3. Detect Smoke	A2-3. Contact Authorities	Alarm Notification

14	A1-3. Detect Smoke	A3. Activate Alarm	Smoke Detection
15	A1-2. Detect Heat	A2-1. Contact Monitoring Service	Alarm Notification
16	A1-2. Detect Heat	A2-3. Contact Authorities	Alarm Notification
17	A1-2. Detect Heat	A3. Activate Alarm	Heat Detection
18	A1-5. Detect Flood	A2-1. Contact Monitoring Service	Flood Notification

Table 10. Final IER List

By examining the IERs from all the functional areas together, we can see that 5, 18, and 23; 26, 11, and 21; 6, 19, and 24; and 7, 20, and 25 are duplicative. Therefore, 8 redundant IERs can be eliminated resulting in the final list of IERs applicable to the domain, found in Table 10:

d. Smartly Aggregate Activities Into Roles/Nodes

At this point, all the activities and information exchange requirements between those activities have been identified. Now, it is necessary to group these activities into roles (or nodes) based on the needs of the functional area.

There are many different considerations to take into account when making these groupings. First, you may want to cluster activities together that would normally be accomplished by the same group, i.e. those activities that make sense to be together. Second, you may want to cluster activities to eliminate the need for information exchanges, i.e. making these exchanges intra-nodal instead of inter-nodal. This approach could reduce risk introduced by system interoperability problems. If redundancy is important in the functional area (as it is in many DoD functional areas,) it may make sense to assign activities to multiple roles. There exists no right or wrong combination of these approaches--so long as by the end of the step, the players/roles within the functional area have been defined and there is a clear assignment of activities (requirements) and information exchange requirements between them.

Continuing with the home security example, the activities within the different functional areas were aggregated as depicted in the following figures:

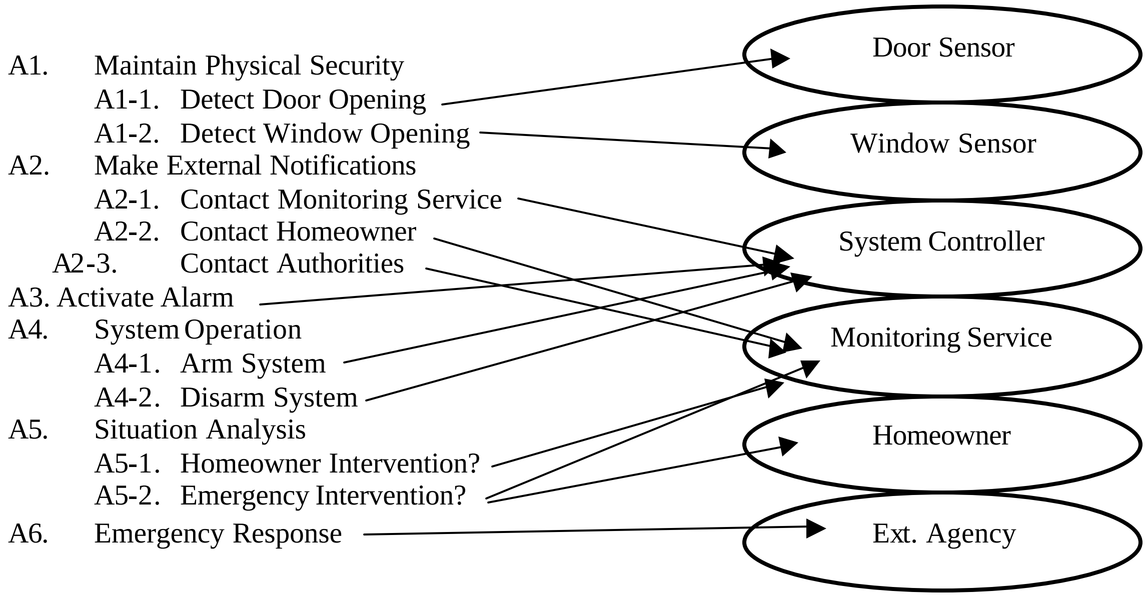

A1. Maintain Physical Security
A1-1. Detect Door Opening
A1-2. Detect Window Opening
A2. Make External Notifications
A2-1. Contact Monitoring Service
A2-2. Contact Homeowner
A2-3. Contact Authorities
A3. Activate Alarm
A4. System Operation
A4-1. Arm System
A4-2. Disarm System
A5. Situation Analysis
A5-1. Homeowner Intervention?
A5-2. Emergency Intervention?
A6. Emergency Response
Door Sensor
Window Sensor
System Controller
Monitoring Service
Homeowner
Ext. Agency

Figure 3. Intruder Detection Roles

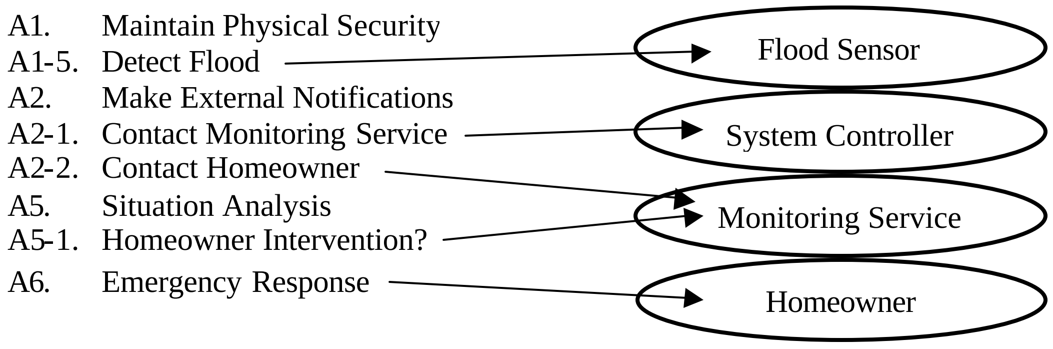


Figure 4. Fire Detection Roles

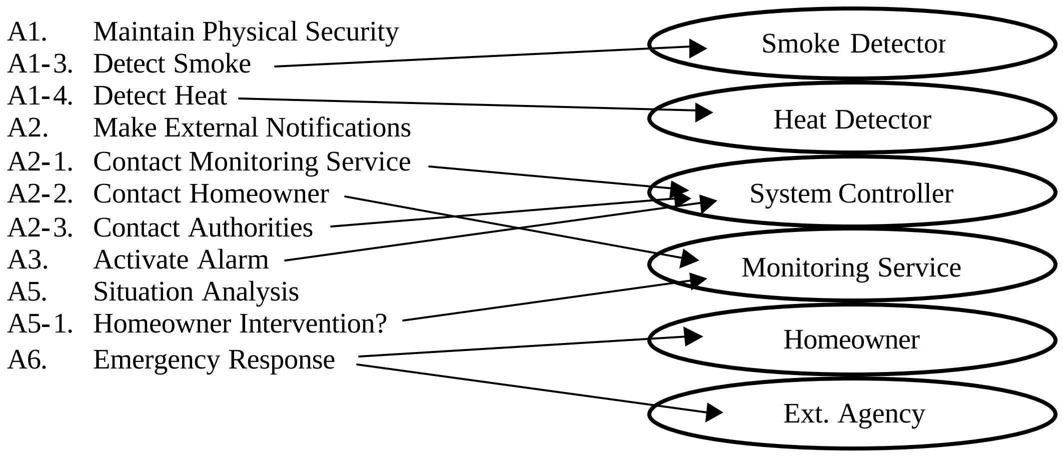


Figure 5. Flood Detection Roles

Once a role is defined and activities assigned to it, the role inherits the Information Exchange Requirements of those activities. It is in this manner that a role is given responsibility for a particular IER. In the post-assignment analysis, it may be found that several of these IERs will disappear from the top-level operational architecture, as both the originating and receiving activities are contained within the same node. By looking at the example of home security, the following "aggregated" IER list (by functional area) was created through such an analysis:

	Originating Role	Originating Activity	Receiving Role	Receiving Activity	Information Element
ID1	Door Sensor	A1-1. Detect Door Opening	System Controller	A3. Activate Alarm	Door Status
ID2	Window Sensor	A1-2. Detect Window Opening	System Controller	A3. Activate Alarm	Window Status
ID3	System Controller	A2-1. Contact Monitoring Service	Monitoring Service	A5-1. Homeowner Intervention Decision	Alarm Notification
ID4	Monitoring Service	A2-2. Contact Homeowner	Homeowner	A5-2. Emergency Intervention Decision	Alarm Notification
ID5	Homeowner	A5-2. Emergency Intervention Decision	Monitoring Service	A2-3. Contact Authorities	Intervention Decision
ID6	Monitoring Service	A2-3. Contact Authorities	Emergency Response Agency	A6. Emergency Response	Alarm Notification

Table 11. Intruder Detection Aggregated IER List

	Originating Role	Originating Activity	Receiving Role	Receiving Activity	Information Element
FD1	Smoke Detector	A1-3. Detect Smoke	System Controller	A3. Activate Alarm	Smoke Detection
FD2	Heat Detector	A1-2. Detect Heat	System Controller	A3. Activate Alarm	Heat Detection

FD3	System Controller	A2-1. Contact Monitoring Service	Monitoring Service	A5-1. Homeowner Intervention Decision	Alarm Notification
FD4	Monitoring Service	A2-2. Contact Homeowner	Homeowner	A6. Emergency Response	Alarm Notification
FD5	System Controller	A2-3. Contact Authorities	Emergency Response Agency	A6. Emergency Response	Alarm Notification

Table 12. Fire Detection Aggregated IER List

	Originating Role	Originating Activity	Receiving Role	Receiving Activity	Information Element
FL1	Flood Sensor	A1-5. Detect Flood	System Controller	A2-1. Contact Monitoring Service	Flood Notification
FL2	System Controller	A2-1. Contact Monitoring Service	Monitoring Service	A5-1. Homeowner Intervention Decision	Alarm Notification
FL3	Monitoring Service	A2-2. Contact Homeowner	Homeowner	A6. Emergency Response	Alarm Notification

Table 13. Flood Detection Aggregated IER List

Through this exercise, we see that the list of 18 previous IERs within the domain has been reduced to 14, thus simplifying the internodal dependencies for the functional area. Depending on how systems are implemented within the domain, these 4 IERs may yet still be system interoperability requirements (some nodes may be made up of multiple systems,) but this aspect of the architecture will be accounted for in later steps.

This particular activity (of defining nodes) gains particular strength when using a data-centric approach to the architecture. In his design of a functional area, the architect may choose only certain activities for certain roles. But because all the activities within his functional area (and the entire domain, for that matter) are visible to the architect at any time, it is relatively easy to expand a role if it becomes necessary later in the architecture's development.

e. Role-Centric Architecture – Operational Perspective

The tasks embedded in Step Two focused on the creation of activities, information exchange requirements and roles. Put together, these objects and the defined relationships between them are the foundation for the Operational Perspective of the Role-centric Architecture. Role-centric architectures are intended to be of greatest to use to the planning community. Requirements developers, users, any stakeholder who has direct impact on the operational requirements of a system gains the most benefit from the information the role-centric architectures contain. To complete the operational portion of the role-centric architecture, activities were defined and hierarchically organized (parent-child activities defined), *Information Exchange Requirements* were identified as need-lines between activities, and *Roles* were defined as groupings of these activities. The beginning of the proposed architectural data model, capturing these relationships, is included in Figure 6.

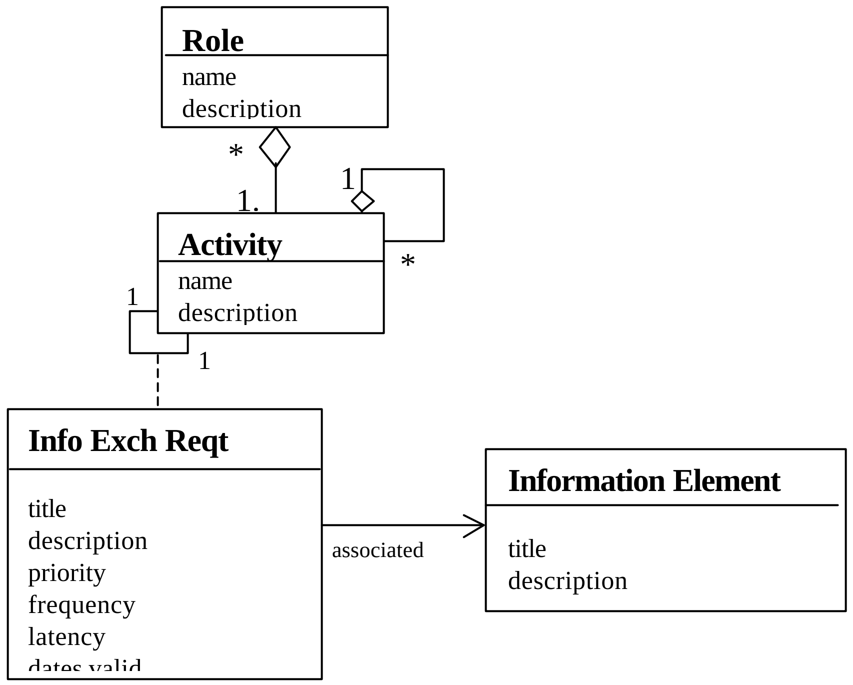


Figure 6. Operational Perspective, Role-Centric Architecture Data Model[3]

[3] Although objects in the model contain attributes, these are provided as an example of the types of attributes that may want to be considered. These may change depending on the focus of the architecture and should not be considered as the only attributes that may be assigned these elements.

To this point, the interoperability requirements for the domain are defined and have been assigned (along with their parent activities) to nodes within the domain. Now it is a matter of assigning these activities and IERs to systems while providing a framework for growth and maintenance.

3. Step Three: Document Your Capabilities

In order to assign activities to systems, it is necessary to first understand the current capabilities of the systems that are available to a given functional area. This involves first documenting current system *capabilities* in the same terms in which operational *activities* were defined. After that step is complete, system *interfaces* are defined in the same terms in which *information exchange requirements* were defined.

Like activities, the architecture supports the concept of embedded systems. A *system* may be stand-alone, or may be part of a system of systems or family of systems. In this way, the architecture also supports viewing systems at a micro-level. A single software application can be broken into its separate modules to show which of those modules perform which activities and where within the application the interface to the external system lies. In this way, the architecture can be used to capture threads and traces within a system.

Additionally, this feature is useful to system integrators who are responsible for fielding a system of integrated components—often seen in the DoD, and often with the integrator having little to no time during software code and development. For instance, the Global Command and Control System is comprised of several software applications, like JOPES, COP, GSORTS, TRANSCOP, and I3, just to name a few. This methodology, if applied to its fullest, can be used to show not only how GCCS interacts with it external systems, but how the components of GCCS interact with each other to complete the requirements of the functional area. Instead of looking at the architecture from the perspective of interoperability, one could easily go a few levels deeper in detail to look at the architecture from the perspective of integration.

Systems with improper documentation may have to be reverse engineered to complete this step. Although reverse engineering can be a costly endeavor, it is vitally important to the remainder of the architecture that a comprehensive understanding of system capabilities is available within the architecture. If a completely new domain is being architected, this step may not be necessary; however, it can be used to conduct market research and document commercial technologies that may be available when it comes time to develop and field systems. The result of this step will be a standardized repository for all system capabilities and interfaces. These results should be made available to all architects within the domain, as system *capabilities* do not change with functional areas.

To continue the storyboarded example, a hypothetical 'market analysis' was completed to determine what systems are available that can meet the objectives defined in the operational perspective of our role-centric architecture. Candidate systems were identified and system capabilities were matched up against the activities defined in the previous step. If a system was capable of achieving this activity, this was documented as a *system function*. If a system was capable of exchanging information with another system, this was documented as a *system information exchange capability* (SIEC). Like their related IERs, SIECs are documented between system pairs and associated with an *information element*. The 'results' of this market survey are included in the tables below:

	System Functions	System Information Exchange Capabilities
Sens1	A1-1, A1-2	Cont1(ID1) Cont1(ID2)
Sens2	A1-1	Cont2(ID1) Cont3(ID1)
Sens3	A1-2	Cont1(ID2) Cont2(ID2)
Sens4	A1-2	Cont3(ID2)
Sens5	A1-3, A1-4	Cont1(FD1) Cont1(FD2)
Sens6	A1-3	Cont1(FD1) Cont2(FD1)
Sens7	A1-5	Cont1(FD1) Cont2(FD1)

Table 14. Sensor Market Survey Results

	System Functions	System Information Exchange Capabilities
Cont1	A4-1, A4-2, A3, A2-3	Sens1(ID1) Sens1(ID2) Sens6(FD1) Sens5(FD1) Sens5(FD2) Sens7(FL1)
Cont2	A4-1, A4-2, A2-1, A2-3	Sens2(ID1) Sens3(ID2) Sens6(FD1) Mon2(ID3) Sens2(ID1) Sens5(FD2) Sens5(FD1)
Cont3	A4-1, A4-2, A2-1, A3	Sens1(ID1) Sens2(ID1) Sens7(FL1) Sens4(ID2) Mon2(ID3) Sens6(FD1)

Table 15. System Controller Market Survey Results

	System Functions	System Information Exchange Capabilities
Mon1	A2-3	Police1(ID6) Fire1(FD5)
Mon2	A2-3	Cont2(ID3) Cont3(ID3) Police2(ID6) Fire2(FD5)

Table 16. Monitoring Service System Market Survey Results

	System Functions	System Information Exchange Capabilities
Police1	A6	Mon1(ID6)
Police2	A6	Mon2(ID6)
Fire1	A6	Mon1(FD5)
Fire2	A6	Mon2(FD5)

Table 17. External Agency System Market Survey Results

	System Functions	System Information Exchange Capabilities
Phone	A5-2, A6	Mon1(ID4) Mon2(ID4) Mon1(ID5) Mon2(ID5) Mon1(FD4) Mon2(FD4) Mon1(FL3) Mon2(FL3)

Table 18. Homeowner System Market Survey Results

There are a few items of particular note as a result of this market survey. First, is to note that while external agency ‘systems’ may not be under the control of the functional area manager, it is important that the capabilities of these systems are also captured, as communication with them is imperative to the completion of the mission. For instance, these systems may be the 911 emergency response system and, perhaps, a direct feed from the security service into a town's police or fire department. Later, when selecting which monitoring service systems to implement for the functional area, it will be important to know which of these systems support the direct feed and which rely on another system—namely the 911 response operators and the phone company.

Second, even communications networks that might be considered global and standard—like the phone system—need to be documented. If activities within the functional area are expected to be reliant on these systems, that reliance needs to be captured. For instance, it is likely that in any of the circumstances in which the homeowner needs to be contacted, that they will first be contacted by phone. Completion of these activities is reliant on that phone and phone system, and, therefore, this needs to be documented within the architecture.

Third, note that each *interface* is documented with two system identifiers and an IER identifier. Furthermore, each interface is listed twice, once under each system endpoint. Even in this very simple example, the more central systems have as many as 8 SIECs associated with them. In larger functional areas with many more systems, this number can grow exponentially. This highlights the need for a centralized database to track these relationships and dependencies. Without it, it would be impossible to maintain consistency and currency within the architecture.

Lastly, note that each system function and interface is defined in terms of the operational architecture established in Step Two. Without this, the information captured in the architecture is virtually useless, as it will become impossible to track capabilities to requirements (especially as the architecture grows.)

a. System-Centric Architecture – Capabilities Perspective

In this step, the focus was on the relationships between systems, system functions and system information exchange capabilities. These objects and associations can be added to the proposed architectural data model as shown in Figure 7.

It is important to note in the data model that while there can be any combinations of relationships between systems and activities, that a system function is a mapping of only one system to one activity. Also, note that while an *IER* was an association between two *activities* described by an *information element*, an *SIEC* is an association between two *system functions* (which are in themselves associations between a *system* and an *activity*) described by and *Information Element.* Therefore, the *SIEC* is an association in which five previously defined objects take part. And, because an *SIEC* has a direct attachment to the *Information Element* and not the *IER*, it is possible to document capabilities that are not driven by a documented requirement. This will not be true on the requirements side of the system-centric architecture.

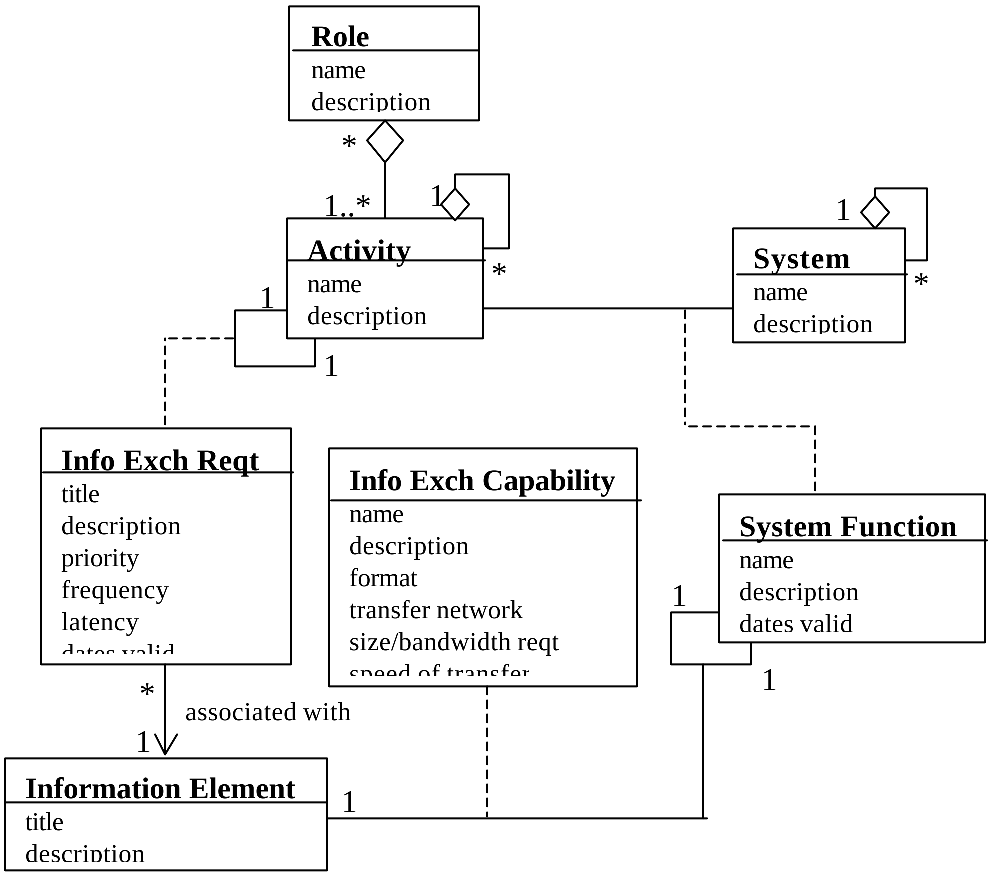


Figure 7. Data Model Relationship between Operational Perspective, Role-Centric and Capabilities Perspective, System-Centric Architectures

In Step Three, the relationship between systems, system functions, and system information exchange capabilities was explained. These objects and their associations collectively are known as the *Capabilities Perspective* of the *System-Centric Architecture*. System-centric architectures are intended to be of greatest use to the developer, integrator, and other stakeholders of the domain that are directly responsible for a systems' implementation. Hence, they are focused upon a particular *system*, as opposed to a functional area.

In the next step, this information will be analyzed to determine which of today's systems can meet the roles that were identified previously in Step Two.

4. Step Four: Determine What Systems are Capable of Meeting What Roles

The next step is to compare the capabilities of today's systems to the activities outlined in the Role-centric architecture to answer two primary questions:

1) Which of today's systems are capable of meeting operational requirements as outlined in the role-centric architecture?
2) Where are there gaps in current capabilities for which new systems or capabilities must be procured?

To complete this step, the activities each system is capable of performing and the interfaces they can support are compared to the activities and IERs that were outlined for each node. Systems are then assigned to nodes based on those comparisons.

It is possible for systems to be assigned to multiple nodes, even within the same functional area. Additionally, multiple systems may be capable of fulfilling any one role and many systems may be tasked to multiple functional areas. For instance, in the functional area of Close Air Support, there is a role called Air Interdictor—a plane that can attack targets on the ground. There are many planes in the current inventory capable of fulfilling this role: A-10, AC-130, F-14, F-15, F-16, F-18, etc. Furthermore, any one of these aircraft could be called in to fulfill a role in another functional area, such as Combat Air Patrol, Air-to-Air Superiority, or Force Protection. This is expected and perfectly acceptable, especially as we acquire more multi-role weapons systems. In fact, it is important to understand all the systems that are capable of fulfilling each role, and all the other systems that could be fielded together to achieve a functional area in order for the full scope of interoperability requirements to be realized. The end result of this step will be an assignment of systems (or groups of systems) to roles and an analysis of what requirements cannot be met by today's systems.

a. Assigning Systems to Roles

In the ongoing example of Home Security, systems were assigned to the roles outlined in the role-centric architecture based on the activities they could perform. For example, Sensor 1 is capable of acting as both a Door Sensor and a Window Sensor, and is, therefore, assigned to both roles. However, Sensor 2 can only be used as a Door Sensor, and so is only assigned to the one role. Using this type of analysis, systems were assigned to roles in the following manner:

	System Functions	System Information Exchange Capabilities	Roles
Sens1	A1-1, A1-2	Cont1(ID1) Cont1(ID2)	ID-Door Sensor ID-Window Sensor
Sens2	A1-1	Cont2(ID1) Cont3(ID1)	ID-Door Sensor
Sens3	A1-2	Cont1(ID2) Cont2(ID2)	ID-Window Sensor
Sens4	A1-2	Cont3(ID2)	ID-Window Sensor
Sens5	A1-3, A1-4	Cont1(FD1) Cont1(FD2)	FD-Smoke Detector FD-Heat Detector
Sens6	A1-3	Cont1(FD1) Cont2(FD1)	FD-Smoke Detector
Sens7	A1-5	Cont1(FL1) Cont3(FL1)	FL-Flood Sensor

Table 19. Sensor Assignments to Roles

	System Functions	System Information Exchange Capabilities	Roles
Cont1	A4-1, A4-2, A3, A2-3	Sens1(ID1) Sens1(ID2) Sens6(FD1) Sens5(FD1) Sens5(FD2) Sens7(FL1)	ID-System Controller FD-System Controller FL-System Controller
Cont2	A4-1, A4-2, A2-1, A2-3	Sens2(ID1) Sens3(ID2) Sens6(FD1) Mon2(ID3) Sens2(ID1) Sens5(FD2) Sens5(FD1)	ID-System Controller FD-System Controller

Cont3	A4-1, A4-2, A2-1, A3	Sens1(ID1) Sens2(ID1) Sens7(FL1)
		Sens4(ID2) Mon2(ID3) Sens6(FD1)
			ID-System Controller
			FD-System Controller
			FL-System Controller

Table 20. System Controller Assignments to Roles

	System Functions	System Information Exchange Capabilities	Roles
Mon1	A2-3	Police1(ID6) Fire1(FD5)	I-Monitoring Service F-Monitoring Service FL-Monitoring Service

Mon2	A2-3	Cont2(ID3) Cont3(ID3)
		Police2(ID6) Fire2(FD5)
			I-Monitoring Service
			F-Monitoring Service
			FL-Monitoring Service

Table 21. Monitoring System Assignments to Roles

In all likelihood, not all the systems will be capable of performing every activity and of supporting every interface. When it comes time to implementing actual combinations of systems, many considerations may affect the final decision. For instance, in the Fire Detection Functional Area, Sensor 5 is only capable of talking to Controller 1 and, likewise, Sensor 6 is only capable of communicating with Controller 2. However, Sensor 5 is also the only sensor capable of sensing heat. It is also important when making these assignments to also remember that a parent system, by definition, brings with it all of its children. Additionally, it is of note that Controller 2 was not assigned to the Flood Detection functional area. Although a 'System Controller' by name (and, therefore, likely to be considered for this role), the analysis revealed that Controller 2 did not support any of the flood detection activities or interfaces and, therefore, was not well suited to that functional area.

There are many areas where tradeoffs will have to be considered when making a final implementation decision. But, this data-driven approach to architecture will provide the necessary information to make those decisions.

b. Role-Centric Architecture – Systems Perspective

This mapping of systems to roles, based on the activities they can perform has provided a link between the Operational Perspective of the Role-Centric Architecture and the Capabilities Perspective of the System-Centric Architecture. (See Figure 8)

The only change to the data model is the adding of an unattributed association between a *system* and a *role,* which is depicted in Appendix A and subsequent views of the model. Additionally, it is important to recognize that there is no difference in the data contained in the Capabilities Perspective of the System-Centric architecture and Figure the Systems Perspective of the Role-Centric Architecture. In fact, the exact subset of data makes up both. The difference is rather in the manner in which the data is viewed and used. In the Capabilities Perspective, the focus is on a single system and all its capabilities regardless of functional area.

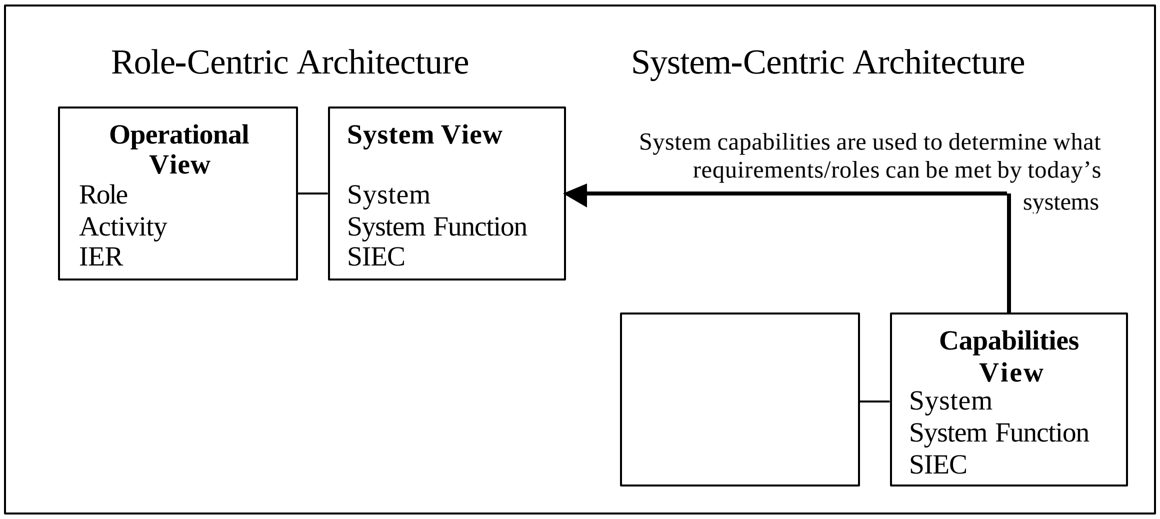

Role-Centric Architecture
System-Centric Architecture
Operational View
Role
Activity
IER
System View
System
System Function
SIEC
System capabilities are used to determine what requirements/roles can be met by today's systems
Capabilities View
System
System Function
SIEC

Figure 8. Relationship between Role-Centric and System-Centric Architectures (Partial)

In the Systems Perspective of the Role-Centric Architecture, the focus is on the functional area and the combined capabilities of all the systems that have been assigned to that functional area.

With systems assigned to roles, the operational requirements of those roles can be levied in turn back onto the systems community to fill in the gaps in capability that were revealed during the analysis.

5. Step Five: Levy Interoperability Requirements on Systems

At this point, the architect has successfully identified the needs of his functional area (in terms of roles, activities, and IERs), the capabilities of available systems (in terms of systems, functions, and SIECs), and done the first part of analysis to determine how his current systems meet the needs of his functional area. It is now necessary to close the loop by determining which of his operational needs could not be met by today's systems and levying those back on system developers in terms of tomorrow's requirements. As has been discussed previously, operational interoperability requirements determine system interoperability requirements [3]. But this determination could not have been made properly without the work of the previous four steps.

To complete this step, the systems will inherit the interoperability requirements of the activities to which they are assigned. This becomes the basis of all future interoperability requirements and testing. Operational *activities* get levied on systems in the form of *system requirements.* And *IERs* get levied on systems in the form of *System Information Exchange Requirements.*

In functional areas where there are multiple systems assigned to roles, it is important to propagate the entire spectrum of combinatory possibilities as requirements. For instance, the role of the Air Interdictor in the Close Air Support functional area was previously discussed. If this role had an IER with another role in the functional area, suppose an Airborne Command Node. Then all the aircraft assigned to the Air Interdictor role (A-10, AC-130, F-14, F-15, F-16, F-18) would be given that IER with all the aircraft assigned to the Airborne Command role (AWACS, P-3, JSTARS), for a total of 18 *SIERs* generated from the 1 *IER.*

Gap analyses can be lengthy and complex. One benefit of the centralized database is that it allows tools to be developed that can accomplish this gap analysis automatically. This step ultimately results in documented operational requirements for all the systems in the domain and, possibly, an identification of need for additional systems.

Continuing with the example of home security, a gap analysis was performed of current system capabilities versus requirements to identify any holes.

	Roles	Sys Reqts	Sys Funcs	SIERs	SIECs

Cont1	ID-System Controller
	FD-System Controller
	FL-System Controller
		A2-1
		A3
		A4-1
		A4-2
		A2-3
			A4-1
			A4-2
			A3
			A2-3
				Sens1(ID1)
				Sens2(ID1)
				Sens1(ID2)
				Sens3(ID2)
				Sens4(ID2)
				Mon1(ID3)
				Mon2(ID3)
				Sens5(FD1)
				Sens6(FD1)
				Sens5(FD2)
				Mon1(FD3)
				Mon2(FD3)
				Fire1(FD5)
				Fire2(FD5)
				Sens7(FL1)
				Mon1(FL2)
				Mon2(FL2)
					Sens1(ID1)
					Sens1(ID2)
					Sens6(FD1)
					Sens5(FD1)
					Sens5(FD2)
					Sens7(FL1)

Cont2	ID-System Controller
	FD-System Controller
		A2-1
		A3
		A4-1
		A4-2
		A2-3
			A4-1
			A4-2
			A2-1
			A2-3
				Sens1(ID1)
				Sens2(ID1)
				Sens1(ID2)
				Sens3(ID2)
				Sens4(ID2)
				Mon1(ID3)
				Mon2(ID3)
				Sens5(FD1)
				Sens6(FD1)
				Sens5(FD2)
				Mon1(FD3)
				Mon2(FD3)
				Fire1(FD5)
				Fire2(FD5)
					Sens2(ID1)
					Sens3(ID2)
					Sens6(FD1)
					Mon2(ID3)
					Sens2(ID1)
					Sens5(FD2)
					Sens5(FD1)

Cont3	ID-System Controller
	FD-System Controller
		A2-1
		A3
			A4-1
			A4-2
				Sens1(ID1)
				Sens2(ID1)
					Sens1(ID1)
					Sens2(ID1)

	FL-System Controller	A4-1
		A4-2
		A2-3
			A2-1
			A3
				Sens1(ID2)
				Sens3(ID2)
				Sens4(ID2)
				Mon1(ID3)
				Mon2(ID3)
				Sens5(FD1)
				Sens6(FD1)
				Sens5(FD2)
				Mon1(FD3)
				Mon2(FD3)
				Fire1(FD5)
				Fire2(FD5)
				Sens7(FL1)
				Mon1(FL2)
				Mon2(FL2)
					Sens7(FL1)
					Sens4(ID2)
					Mon2(ID3)
					Sens6(FD1)

Table 22. System Controller Gap Analysis

	Roles	System Requirements	System Functions	System Information Exchange Requirements	System Information Exchange Capabilities

Sens1	ID-Door Sensor
	ID-Window Sensor
		A1-1
		A1-2
			A1-1
			A1-2
				Cont1(ID1)
				Cont1(ID2)
				Cont2(ID1)
				Cont2(ID2)
				Cont3(ID1)
				Cont3(ID2)
					Cont1(ID1)
					Cont1(ID2)

Sens2	ID-Door Sensor	A1-1	A1-1	*Cont1(ID1)*
				Cont2(ID1)
				Cont3(ID1)
					Cont2(ID1)
					Cont3(ID1)

Sens3	ID-Window Sensor	A1-2	A1-2	Cont1(ID2)
				Cont2(ID2)
				Cont3(ID2)
					Cont1(ID2)
					Cont2(ID2)

Sens4	ID-Window Sensor	A1-2	A1-2	*Cont1(ID2)*
				Cont2(ID2)
				Cont3(ID2)
					Cont3(ID2)

Sens5	FD-Smoke Detector
	FD-Heat Detector
		A1-3
		A1-4
			A1-3
			A1-4
				Cont1(FD1)
				Cont1(FD2)
				Cont2(FD1)
				Cont2(FD2)
					Cont1(FD1)
					Cont1(FD2)

				Cont3(FD1)
				Cont3(FD2)	

Sens6	FD-Smoke Detector	A1-3	A1-3	Cont1(FD1)
				Cont2(FD1)
				Cont3(FD1)
					Cont1(FD1)
					Cont2(FD1)

Sens7	FL-Flood Sensor	A1-5	A1-5	Cont1(FL1)
				Cont3(FL1)
					Cont1(FL1)
					Cont3(FL1)

Table 23. Sensor Gap Analysis

	Roles	System Requirements	System Functions	SIERs	SIECs

Mon1	ID-Monitoring Service
	FD-Monitoring Service
	FL-Monitoring Service
		A2-2
		A2-3
		A5-1
		A5-2
			A2-3	*Cont1(ID3)*
				Cont2(ID3)
				Cont3(ID3)
				Phone(ID4)
				Phone(ID5)
				Police1(ID6)
				Police2(ID6)
				Cont1(FD3)
				Cont2(FD3)
				Cont3(FD3)
				Phone(FD4)
				Cont1(FL2)
				Cont3(FL2)
				Phone(FL3)
					Police1(ID6)
					Fire1(FD5)

Mon2	ID-Monitoring Service
	FD-Monitoring Service
	FL-Monitoring Service
		A2-2
		A2-3
		A5-1
		A5-2
			A2-3	*Cont1(ID3)*
				Cont2(ID3)
				Cont3(ID3)
				Phone(ID4)
				Phone(ID5)
				Police1(ID6)
				Police2(ID6)
				Cont1(FD3)
				Cont2(FD3)
				Cont3(FD3)
				Phone(FD4)
				Cont1(FL2)
				Cont3(FL2)
				Phone(FL3)
					Cont2(ID3)
					Cont3(ID3)
					Police2(ID6)
					Fire2(FD5)

Table 24. Monitoring System Gap Analysis

There are many observations that can be made as a result of this gap analysis. In most cases, the analysis has shown holes between our requirements and capabilities that can be levied as future requirements on our systems. However, there are a few exceptions:

1) In the case of the monitoring service systems, it is noted there are several activities that are currently not being met by the system capabilities. However, upon further examination, activities 5-1 and 5-2 are decision-making activities to be accomplished by the *role* of monitoring service, and are not necessarily system activities. These need not be levied on the systems for future development unless it is desired that the systems start making these decisions in the future (which may or may not be likely.)

2) Also looking at the monitoring service systems, there are SIECs listed that do not match up against any known SIERs. For instance, the ability to communicate with the Fire Department is desired at the System Controller level. However, this is an existing capability regardless of requirement. Therefore, it is perfectly valid for it to be documented in the architecture. And, in this case, would be useful for the architect to know as none of his system controllers are currently capable of completing this task. (It may be possible to re-route this information through the monitoring service as an interim until a system fix can be made at the controller level.) This, again, highlights one of the advantages of this approach to systems requirements and tracking.

3) With regards to the sensors, the analysis shows that while the sensors meet all their functional requirements, they lack severely in the interoperability capabilities. This is the intended outcome of this methodology—to clearly show where there are systems currently in the field, conducting tasks, that do not meet the interoperability needs of today's users.

The solution is simple: the DoD needs a structured methodology by which to define these interoperability requirements, levy them on systems and track them through implementation.

Through this last step, the feedback between operational requirements and system requirements is complete. Using the information gathered regarding the capabilities of today's systems, a thorough understanding of where system shortfalls lie, and where requirements are lacking has been gained. The relationship between the Role-Centric and System-Centric architectures is complete.

The mapping of activities and IERs back onto systems has resulted in two additions to the data model: *system requirements* and *system information exchange requirements*. Similar to a *system function*, the *system requirement* is an association between an *activity* and a *system*. Likewise, an *SIER* is a mapping between and *IER* and two *system requirements*. The completed data model is located in Appendix A.

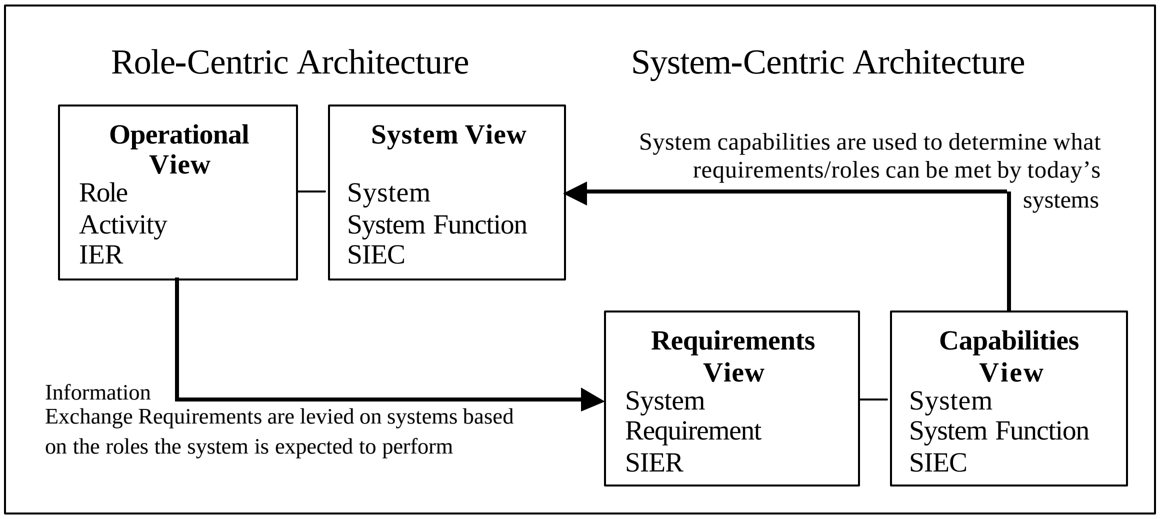

Role-Centric Architecture
System-Centric Architecture
Operational View
Role
Activity
IER
System View
System
System Function
SIEC
System capabilities are used to determine what requirements/roles can be met by today's systems
Requirements View
System
Requirement
SIER
Capabilities View
System
System Function
SIEC
Information Exchange Requirements are levied on systems based on the roles the system is expected to perform

Figure 9. Relationship between Role-Centric and System-Centric Architecture (Complete)

6. Step Six: Prepare for the Future

The final step of this methodology is a continuous activity. These proposed architectures and methods represent living representations of the domain and should maintained over time to reap optimal benefits from the architectures. But, in order to accomplish this, the elements of the architecture must be attributed to facilitate the concept of time. Through time-phasing, the architecture can be adapted to document future requirements, future and planned capabilities, and can help align software systems that are in orthogonal spirals of the spiral development process.

a. Time-Phasing Requirements

Each element of the proposed architecture has the ability to be attributed with a period of time over which it is valid. Requirements generally will not change as often as systems do, especially in the defense world, as it has often been argued that new technologies do not fundamentally change the way we fight wars. However, new initiatives like Joint Vision 2010, Joint Vision 2020, C4I for the Warrior, and other programs that look at how we can change the way we fight based on the capabilities of today's systems drive us towards new requirements for our systems of the future.

These requirements can be captured in the same architecture used to document today's requirements by using a date stamp on each architectural element. In this way, system developers will be able to anticipate the requirements of their systems one, two, or even 10 to 20 years down the line. One suggestion for the defense and government sectors is that requirements be adjusted around the POM cycle—a five-year cycle through which future budgets are planned. This would provide a mechanism to better align programmatic dollars where the requirements are going to be.

In the process of time-phasing requirements, there are two types of requirements that must be reviewed. First are the operational requirements of the domain or functional area—those that were captured in the *operational* perspective of the *role-centric architecture*. When phasing these requirements, the architect would approach for a more 'visionary' stance. *How should this functional area be conducted in 5 years? How can we better align these activities to fight the wars of the future? What benefits can I reap from the systems that are coming down line to further decentralize my execution and eliminate multiple command and communication nodes?* These are the types of considerations that can be made when looking at the role-centric architectures of tomorrow.

There are also those operational requirements as they were levied on the systems, in the forms of *system requirements* and *SIERs*. It is unlikely that the system requirements will have the same lifespan as their operational parents; again as our systems are likely to change more often than the way we conduct current operations. When phasing these requirements, the architect must take an approach that mixes ‘vision’ with ‘reality.’ For instance, the architect might decide he can better serve a functional area with the introduction of a new multi-role fighter in 5 to 7 years—one that takes on many of the activities outlined in the architecture to reduce the number of fielded systems. This is an example of more visionary thinking. However, the architect must also consider that the new Joint Strike Fighter (JSF), for instance, is going to replace other aging aircraft; so requirements that used to be on the F-16 and F-18, for instance, need to be moved to the JSF at the appropriate time in the future. This is an example of reality-based phasing.

Incorporating a time-phased approach to requirements into part of the Home Security example, we might decide that in the Intruder Detection Functional Area, that today’s requirements of door and window intrusion detection are sufficient for today; but, in ten years it will be necessary to also detect wireless cyber-intrusion. In that case, the operational requirements of the functional area would be updated to include a new activity with all the necessary IERs.

	Description	Dates Valid
A1.	Maintain Physical Security	200201 - 202201
A1-1.	Detect Door Opening	200201 - 202201
A1-2.	Detect Window Opening	200201 - 202201
A1-3.	Detect Smoke	200201 - 202201
A1-4.	Detect Heat	200201 - 202201
A1-5.	Detect Flood	200201 - 202201
A1-6.	*Detect Cyber-Intrusion*	*201201 - 202201*
A2.	Make External Notifications	200201 - 202201
A2-1.	Contact Monitoring Service	200201 - 202201
A2-2.	Contact Homeowner	200201 - 202201
A2-3.	Contact Police	200201 - 202201
A2-4.	Contact Fire Department	200201 - 202201
A3.	Activate Alarm	200201 - 202201
A4.	System Operation	200201 - 202201
A4-1.	Arm System	200201 - 202201
A4-2.	Disarm System	200201 - 202201
A5.	Situation Analysis/Decision Point	200201 - 202201

A5-1.	Decide if Homeowner Intervention is Required	200201 - 202201
A5-2.	Decide if Emergency Response is Required	200201 - 202201
A6.	Emergency Response	200201 - 202201
A6-1.	Investigate/Respond to Security Alarm	200201 - 202201
A6-2.	Respond to Fire Alarm	200201 - 202201

Table 25. Updated Activity List

	Originating Role	Originating Activity	Receiving Role	Receiving Activity	Information Element	Dates Valid
ID 1	Door Sensor	A1-1. Detect Door Opening	System Controller	A3. Activate Alarm	Door Status	200201 – 202201
ID 2	Window Sensor	A1-2. Detect Window Opening	System Controller	A3. Activate Alarm	Window Status	200201 – 202201
ID 3	System Controller	A2-1. Contact Monitoring Service	Monitoring Service	A5-1. Homeowner Intervention Decision	Alarm Notification	200201 – 202201
ID 4	Monitoring Service	A2-2. Contact Homeowner	Homeowner	A5-2. Emergency Intervention Decision	Alarm Notification	200201 – 202201
ID 5	Homeowner	A5-2. Emergency Intervention Decision	Monitoring Service	A2-3. Contact Authorities	Intervention Decision	200201 – 202201
ID 6	Monitoring Service	A2-3. Contact Authorities	Emergency Response Agency	A6. Emergency Response	Alarm Notification	200201 – 202201
ID 7	*Cyber Sensor*	*A1-7. Detect Cyber-Intrusion*	*System Controller*	*A3. Activate Alarm*	*Cyber Attack Notification*	*201201 – 202201*
ID 8	*System Controller*	*A2-1. Contact Monitoring Service*	*Monitoring Service*	*A5-1. Homeowner Intervention Decision*	*Cyber Attack Notification*	*201201 – 202201*

Table 26. Updated Intruder Detection IER List

Following the update within the role-centric architecture, it is necessary to ensure that the changes are considered for the system-centric architecture, as well. It is assumed that the current door and window sensors are not well suited to detect cyber attacks, and therefore, it will be necessary to identify a requirement in the architecture for a new start system and correctly assign the system requirements and SIERs, accordingly. In this case, the architect has chosen to also review the time stamps of current system requirements. The result follows:

	Roles	System Reqts	Dates Valid	SIERs	Dates Valid

Sens1	ID-Door Sensor
	ID-Window Sensor
		A1-1
		A1-2
			200201 – 202201
			200201 - 202201
				Cont1(ID1)
				Cont1(ID2)
				Cont2(ID1)
				Cont2(ID2)
				Cont3(ID1)
				Cont3(ID2)
					200201 – 202201
					200201 – 202201
					200407 – 202201
					200407 – 202201
					200201 – 201010
					200201 – 201010

Sens2	ID-Door Sensor	A1-1	200201 - 202201	Cont1(ID1)
				Cont2(ID1)
				Cont3(ID1)
					200201 – 202201
					200407 – 202201
					200201 – 201010

Sens3	ID-Window Sensor	A1-2	200201 - 202201	Cont1(ID2)
				Cont2(ID2)
				Cont3(ID2)
					200201 – 202201
					200407 – 202201
					200201 – 201010

Sens4	ID-Window Sensor	A1-2	200201 - 202201	Cont1(ID2)
				Cont2(ID2)
				Cont3(ID2)
					200201 – 202201
					200407 – 202201
					200201 – 20101

Sens5	FD-Smoke Detector
	FD-Heat Detector
		A1-3
		A1-4
			200201 – 202201
			200201 – 202201
				Cont1(FD1)
				Cont1(FD2)
				Cont2(FD1)
				Cont2(FD2)
				Cont3(FD1)
				Cont3(FD2)
					200201 – 202201
					200201 – 202201
					200407 – 202201
					200407 – 202201
					200201 – 201010
					200201 – 201010

Sens6	FD-Smoke Detector	A1-3	200201 - 202201	Cont1(FD1)
				Cont2(FD1)
				Cont3(FD1)
					200201 – 202201
					200407 – 202201
					200201 – 201010

Sens7	FL-Flood Sensor	A1-5	200201 - 202201	Cont1(FL1)
				Cont3(FL1)
					200201 – 202201
					200201 – 201010

New	*ID-Cyber Sensor*	*A1-6*	*201201 - 202201*	*Cont1(ID7)*
				Cont2(ID7)
					201201 – 202201
					201201 – 202201

Table 27. Sensor Systems with Time-Phased Requirements

In this scenario, the new 'cyber-sensor' requirements were added to the system-centric architecture under the system name 'New,' to clearly point out the emerging requirement. Because the system is not expected to be out for another 10 years, the dates and the previously assigned activities and IERs were adjusted accordingly. Additionally, a brief analysis of the System Controller situation revealed that while Controller 1 was available for the lifetime of the architecture (Jan 2002 – Jan 2022), Controller 2 would not be out on the market until July 2004 and Controller 3 was anticipated to go End-of-Life in October 2010. With this in mind, it was determined there was no need to establish a requirement for the Cyber-Sensor to interoperate with Controller 3 as the controller would no longer be available by the time the cyber-sensor was ready to be fielded. The *IER* remains valid over the entire lifetime of the architecture, as the requirement itself has not changed. However, when this is mapped to a specific system (as an *SIER*,) the dates may change depending on the lifetime and maturity of that system.

This is just one example of the power time-stamping architectural elements can yield. To be able to capture and maintain this kind of information is to provide the tools to see the evolution of requirements over time and greatly facilitate better decision-making.

b. Time-Phasing Capabilities

These lessons can also be applied to the development community. Software systems, more than any other fielded technologies, are vulnerable to change and changing requirements. The current DoD acquisition lifecycle expects software systems to update every 18 months, and the spiral development cycle is pushing that down to 6. Regardless, as systems are put on contract, future requirements are typically not well understood, and as a result, cost overruns due to poor requirements definition abound. As the DoD moves forward with time-phasing requirements, a methodology must be in place for the acquisition community to also incorporate these concepts into the development cycle, and be able to feed back their progress to the operators and requirements community.

Time-phasing capabilities can provide two key pieces of information: when a capability will be available, and when it is going away. The latter is especially important to help sustain current capabilities. As discussed previously, interoperability is a state. And being able to track that condition of that state over time is imperative. By knowing when systems are going to be discontinued, or interfaces no longer supported, that state of interoperability can be maintained more easily.

The example of home security continues to show how time-phasing capabilities is worked into the architecture. The below table shows how different systems, despite the fact that they may have a valid requirement over an extended period of time, may have to implement this requirement into capability differently. This information is vital to those planners determining what systems are to be fielded, especially in deciding sets of interoperable systems that must be deployed.

	Roles	SIERs	Dates Valid	SIECs	Dates Valid
Sens1	ID-Door	Cont1(ID1)	200201 – 202201	Cont1(ID1)	200201-202201
	Sensor	Cont1(ID2)	200201 – 202201	Cont1(ID2)	200201-202201
	ID-Window	Cont2(ID1)	200407 – 202201	Cont2(ID1)	200407-202201
	Sensor	Cont2(ID2)	200407 – 202201	Cont2(ID2)	200407-202201
		Cont3(ID1)	200201 – 201010	Cont3(ID1)	200601-201010
		Cont3(ID2)	200201 – 201010	Cont3(ID2)	200601-201010
Sens2	ID-Door	Cont1(ID1)	200201 – 202201	Cont1(ID1)	200509-201105
	Sensor	Cont2(ID1)	200407 – 202201	Cont2(ID1)	200407-201105
		Cont3(ID1)	200201 – 201010	Cont3(ID1)	200201-201010
Sens3	ID-Window	Cont1(ID2)	200201 – 202201	Cont1(ID2)	200201-202201
	Sensor	Cont2(ID2)	200407 – 202201	Cont2(ID2)	200401-202201
		Cont3(ID2)	200201 – 201010		
Sens4	ID-Window	Cont1(ID2)	200201 – 202201	Cont1(ID2)	200301-202201
	Sensor	Cont2(ID2)	200407 – 202201		
		Cont3(ID2)	200201 – 201010	Cont3(ID2)	200201-201010

Sens5	FD-Smoke	Cont1(FD1)	200201 – 202201	Cont1(FD1)	200201-201011
	Detector	Cont1(FD2)	200201 – 202201	Cont1(FD2)	200201-201011
	FD-Heat	Cont2(FD1)	200407 – 202201	Cont2(FD1)	200601-202201
	Detector	Cont2(FD2)	200407 – 202201	Cont2(FD2)	200601-202201
		Cont3(FD1)	200201 – 201010	Cont3(FD1)	200201-201010
		Cont3(FD2)	200201 – 201010	Cont3(FD2)	200201-201010
Sens6	FD-Smoke	Cont1(FD1)	200201 – 202201	Cont1(FD1)	200201-202201
	Detector	Cont2(FD1)	200407 – 202201	Cont2(FD1)	200407-202201
		Cont3(FD1)	200201 – 201010		
Sens7	FL-Flood	Cont1(FL1)	200201 – 202201	Cont1(FL1)	200201-202201
	Sensor	Cont3(FL1)	200201 – 201010	Cont3(FL1)	200201-201010
New	ID-Cyber	Cont1(ID7)	201201 – 202201	Cont1(ID7)	201201-202201
	Sensor	Cont2(ID7)	201201 – 202201	Cont2(ID7)	201201-202201

Table 28. Sensor Systems with Time-Phased Capabilities

The time-phased requirements reveal the development plans of the various system developers and provide planners that additional piece of information to make an informed decision about which systems will be available to the warfighter when they deploy at various times in the future. For instance, this attribute reveals that Sensor 2 is going off the market in May 2011. Any reliance on this sensor after that point would have to be compensated. In addition, the different sensors implement the required interfaces at different times. Some have no intentions of implementing some of the interfaces at any time. At any one future date, a different set of sensor-controller pairs might be needed to complete the needs of the functional area.

These scenarios are not only plausible, they are extremely realistic, especially as the DoD grows more reliant on Commercial-Off-the-Shelf (COTS) products in which the DoD may not be the primary customer. The DoD cannot assume that all its requirements will be implemented in a COTS-based world. Commercial industry has become the forcing function for new technologies, and many vendors will choose to drive with it, rather than step backwards to meet defense needs.

Overall, it is the responsibility of the system developer to maintain the data on their systems. And, therefore, the onus is put on the acquisition community to determine when a capability will be available and how it will be supported. If it is decided that a system will be replaced, this process facilitates a smoother transition from the old system to the new by allowing for easy conversion of requirements and previously documented interfaces. Today, new systems often do not have the luxury of obtaining documentation on the systems they are intended to replace. This methodology allays many of those issues.

c. Satisfying Spiral Development Needs

DoDD 5000.2 mandates "software development and integration shall follow an iterative spiral development process in which continually expanding software versions are based on learning from earlier development." [16] As a result, there are literally thousands of disparate software development projects in the DoD today, each spiraling at their own rates, all of which must be interoperable with at least one or more of the others, and most of which must be integrated into a larger system to be fielded. This creates a constantly changing environment in which capabilities are continuously fielded, with little to no mechanism outside each individual program office to track their implementations. And there is nothing that provides the individual program manager enough information to know if he or she is going to encounter problems with the other systems he is required to interact with as each spiral develops.

The popularity of spiral development cycles is one of the modern changes that make this type of architectural methodology so necessary. Keeping track of emerging and falling capabilities can often be difficult within a program office, much less keeping track of the orthogonal spirals of several systems, all of which are attempting to be interoperable. By time-phasing capabilities in a central repository, it will be easier to determine when information exchange capabilities will come on line and give program offices the opportunity to collaborate their different schedules in a centralized fashion.

As such, the proposed methodology provides a foundation for meaningful communication not only *between* the requirements and development communities, but *within* these communities as well, so program managers will have insight into the activities of the other programs with which he must interact. This quality is of most importance to those responsible for system integration.

Currently, many system integrators are handed (particularly) software applications to be integrated into their systems with little to no knowledge of what that application does, how it will interact with the other components of their system, and how it affects the requirements of their system as a whole. Many of them are not even provided a consolidated set of requirements for the integrated system, and rather rely on the requirements of its components to derive the requirements of the whole. Oftentimes, applications are handed to them with little to no forewarning, on constantly changing schedules, to be integrated immediately to meet a warfighting need. To have a tool through which the integrator could track the separate spiral development cycles of each of his component systems, that provides information on which capabilities are supported in which spiral, and also keeps track of interoperability requirements for his integrated system and how those are met by each of the component systems, would be absolutely invaluable to those responsible for system integration.

d. Resource Planning

Analysis of the completed architecture can aid many other stakeholders besides the requirements and development communities. One such area that could benefit from the architectural information is the resource planning community.

As discussed previously, time-phasing of requirements provides additional information to those responsible for allocating resources and money to systems. By having access to the 'bigger picture' of development activities and customer priorities (requirements can be attributed with a priority, as well) planners can make better informed decisions about where money should be allocated to support the needs of tomorrow's warfighter.

Also, the warplanners of today would benefit from this architectural information, as well. Today's planners face the daunting task of determining which systems should be sent to battle with our soldiers, sailors, and airmen. There currently exists few tools to help them with this task, and as a result, the logistical requirements to send troops to global 'hot spots' is staggering. Redundant systems and capabilities are fielded. Systems that cannot communicate with each other and sent out to the field while systems that are interoperable stay at home--all because there exists no methodology for making informed decisions.

However, using the proposed architectural methodology, this analysis is already accomplished for the warplanner. SIECs reveal which systems can be fielded together and where the gaps will be if one system must be chosen over another for external constraints (e.g. logistical, availability, fiscal, training, etc.)

For instance, in the home security example, if it was determined that an overseas operation required an intruder detection capability, the warplanner could turn to the Intruder Detection functional area to find what the requirements were to complete that operation and what systems were capable of performing it. A fairly simple analysis would reveal that Controller 2 would not be available until July 2004, and if it were determined that the units available to deploy had never been trained on Controller 1, it would be necessary to deploy Controller 3. But, to detect window openings, Controller 3 must be deployed with Sensor 4. If only Sensor 5 was available, the planner would have to make a decision as to whether to not support the "Detect Window Opening" activity, or whether to deploy Controller 1, which does support Sensor 5, and make arrangements for in-theater training.

These are the kinds of decisions our planners need to make every day. Unfortunately, they are currently not provided the necessary tools and information to make the best decisions to support the warfighter. The proposed methodology counters that by providing the required data in a format that allows the planners to make informed decisions about where and how to employ our systems.

Architectures are living projects, to be maintained over time. Although this thesis outlines a Six-Step Approach to creating an architecture, the work is hardly finished there. Any or all of these steps can, and should, be reaccomplished as the environment, requirements, or systems change to ensure that the most accurate, consistent information is available to all the stakeholders at any time.

THIS PAGE LEFT INTENTIONALLY BLANK

V. BENEFITS OF THE DATA-DRIVEN APPROACH

Many benefits of the proposed approach to enterprise architecture have been discussed in other parts of this thesis. Some of these were directed towards use in the Department of Defense and specifically toward systems engineering. In the following pages, some of the more general benefits of this approach towards enterprise architecture are discussed. These benefits apply to any user of a data-driven architectural approach, be they a definer of requirements, a systems integrator, a software developer, or a program manager.

A. MAINTENANCE

A data-driven approach to architecture results in an architecture that is easier to maintain and update compared to its paper or picture-oriented peers. Consider a mature architectural effort, populated with activities, systems, requirements, capabilities, and all the information exchanges. As seen in Appendix A, the data elements are extremely dependent on each other. If one activity must be renamed, or deleted, it can affect literally hundreds or thousands of other architectural elements, depending on the scope of the effort. In a paper-based architecture, or even one captured in a non-relational database or spreadsheet, finding all the links affected by that one changed activity can be extremely difficult.

Take, for instance, the home security architecture outlined in Chapter IV. This architecture is fairly simple, with few data elements and associations. If it were necessary to rename the activity "Notify Monitoring Service" to "Alert Central Office," using a relational database this change would be a simple to change to one field in the table that contains the listing of current activities. However, if this were a current architectural effort, working off another media, such as PowerPoint pictures or non-relational spreadsheets, this one change would need to be propagated through 6 architectures (the role-centric and system-centric architectures could not be dynamically linked as the methodology proposes, nor could the 3 functional areas be contained within a single architecture), changing a total of 3 activities, 8 IERs, 3 roles, 9 systems, 12 SIECs, 9 system requirements, 6 system functions, and 96 SIERs.

This one change in a relational database would require a total of 146 separate changes to keep an architecture that was not data-driven current.

With this, an architecture captured in a database is not only easier to maintain, but is also cheaper. Coordination of paper copies of the architectural views is timely and expensive. Currently, ORD reviews in the DoD can take as long as 6-12 months, sometimes longer, depending on the size and complexity of the ORD and the scope of the changes. Propagation of even simple changes through paper documents is time-consuming and redirects valuable man-hours from more important tasks. Using a data-driven approach, changes can be coordinated electronically using relatively simple tools to control and manage updates.

B. DATA CONFIGURATION, CONTROL, AND CONSISTENCY

Similarly, it is easier to control the architectural data elements using the proposed methodology, especially compared with the current practices of today. Configuration control of architectural elements can be built into the database design, only allowing certain users to make certain types of changes and allowing the architects to retain ownership of their perspective data elements. Also, disparate architectural efforts often face the problem of different architects using different language to capture requirements and IERs. By incorporating a centralized database into the domain's architectural efforts, key data elements, such as activities and systems—those elements upon which the majority of the architecture is based—can be accessed through picklists or other uneditable user interfaces, forcing the architecture to be based to a common language or data standard.

Furthermore, because the architectural elements are contained in only one location, many problems with data consistency are alleviated. A high level of consistency is required within any architecture to maintain usability. As discussed previously, current architectural efforts lack the visibility between efforts to ensure that IERs are documented by both endpoints, i.e. if TBMCS lists an IER with GCCS, does GCCS list the same IER in reverse. However, using the outlined approach, this level of consistency is maintained by forcing all architectural efforts into a common database, thus providing a structured methodology for preventing these kinds of oversights.

C. INTEGRATING MULTIPLE ARCHITECTURES

A data-driven approach to architecture design facilitates the integration of multiple architectures much faster than any picture-based efforts. As seen in the maintainability example, a single activity touched nearly 150 other architectural data elements. In a database, it is a fairly simple task to discover which other objects and associations were tied to that task. In picture-based architectures, there are no such mappings. Every element of the subject architecture would have to be examined during every integration effort to see where the links and dependencies lie.

D. MULTIPLE VIEW CAPABILITIES

By focusing on the data rather than specific views of the data, the proposed methodology leaves considerable flexibility in how the data will be presented to the end user. There exist a variety of tools that can provide access to the data in tabular, database, or even picture format. Being able to view architectural elements in picture format is a surprisingly effective style for representing the data—as long as the data drives the picture and not the other way around. Several modern architectural tools use a graphic user interface to dynamically generate architectural drawings based on the underlying data. Furthermore, users can manipulate the drawings and the changes are passed down to the core data elements. This approach combines the simplicity of the pictures with the strength of the relational database and is an excellent approach to architectural development.

1. Role-Centric vs. System-Centric

The most obvious viewing benefit introduced with this approach is the ability to view the architecture data from either a role-centric or system-centric perspective. Functional area experts may want to view how different systems have been tasked to meet the needs of his functional area while system developers may want to view how their particular systems have been tasked across the many functional areas. These two ‘architectures’ are built off the same set of underlying data. And, therefore, the data can be viewed in any manner the user wishes.

Depending on the power of the tools used to interface with the database, a user could choose to view the interactions between all Command and Control systems assigned to any functional area, or how a specific system, like GCCS, fits into the functional area of Focused Logistics. Similarly, a user could view how the Air Superiority functional area has been developed; or rather view just the Navy components of Theater Air and Missile Defense. A data-driven approach does not constrain the users ability to present the information contained in the architecture, and therefore, is one of the greatest benefits of this methodology.

2. “Roll-Up”/Zoom In/Zoom Out

Storing architectural information within a database also allows the user to choose the level of detail at which to view the architecture. As the GCCS program manager, I might want to see how GCCS is expected to interact with other systems in the many different functional areas. Using a data-driven approach, I could then drill down into the GCCS system itself to view how each of the components of GCCS come together to meet these requirements. The level of detail to be viewed would only be constrained by the amount of information that had been put into the architecture. And, if GCCS were part of a bigger system, like the Air Operations Center, I could ‘zoom out’ to see how my system fit into the bigger picture of the functional area.

3. Time-Phasing

The time-phasing of requirements and capabilities within the architecture enables another viewing benefit—the ability to view the architectural elements at any date in time. As a functional area architect, I could view how systems could best support my functional area today, in 2005, 2010, and 2020 by merely selecting the date I wanted to see. The data elements would then be extracted based on the time attributes they previously had been assigned.

Current architectural efforts often discuss the difference between “as-is” and “to-be” architectures. Unfortunately, because of the discussed shortcomings in the process, these become separate architecture efforts, each demonstrating a single snapshot in time. As soon as the “as-is” architecture is developed, it is out of date because system capabilities have changed. And the “to-be” architecture can only be developed to a single future date, often 5 or 10 years in the future. But neither provides the flexibility to ask, “How would the architecture look somewhere in between?” A data-driven approach provides that capability.

THIS PAGE LEFT INTENTIONALLY BLANK

VI. CLOSING COMMENTS

A. RECOMMENDATIONS

The proposed methodologies have a number of applications: defense, commercial and academic; systems engineering, integration, and development; software design, integration, and development. The author recommends their use in any environment which employs distributed systems and transactions to achieve a common purpose.

It is recognized that in many communities, the Department of Defense in particular, changing current processes to incorporate some of these methods is a cumbersome effort. However, the DoD cannot afford to continually throw money at disparate architecture efforts that have no chance of enabling a state of interoperability, as they are uncoordinated, inconsistent and unavailable to the other stakeholders whose participation is vital in a system's development. Only through strong leadership, and a 'top-down' enforcement of disciplined architectural methods will interoperability be achieved.

B. CONCLUSIONS

The use of enterprise architecture techniques in the planning and requirements phases of system development can facilitate system-to-system and software-to-software interoperability. In order to realize the benefits of the architectural methodologies, a structured framework for implementing and maintaining the architecture is required. It must be fully integrated into each stakeholder's processes before full benefit can be realized. The time and resources saved using an integrated, data-driven architectural approach will create a more efficient environment for the development of systems and their requirements, and will enable a state of system interoperability to be achieved and maintained indefinitely.

THIS PAGE LEFT INTENTIONALLY BLANK

APPENDIX A

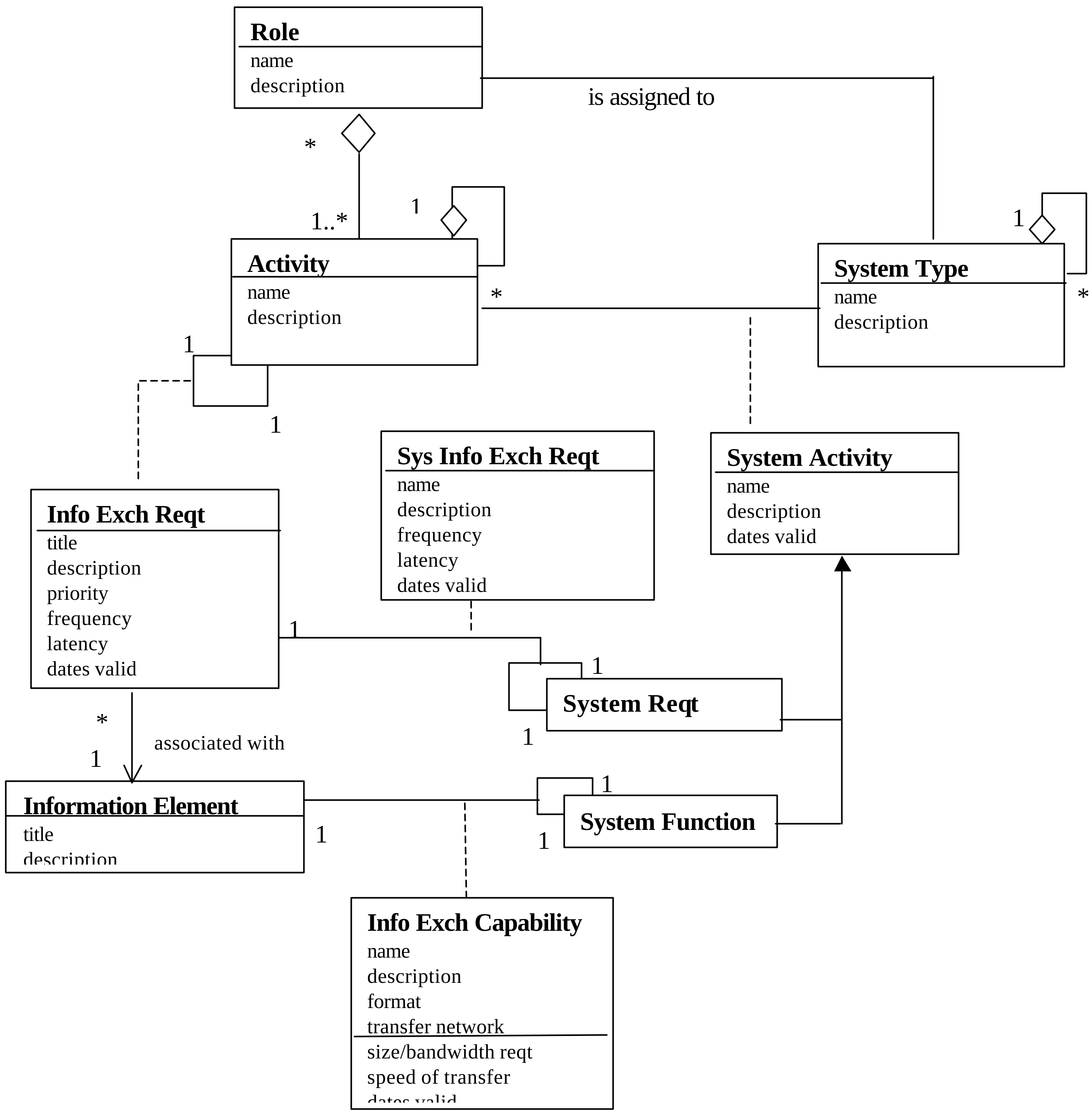
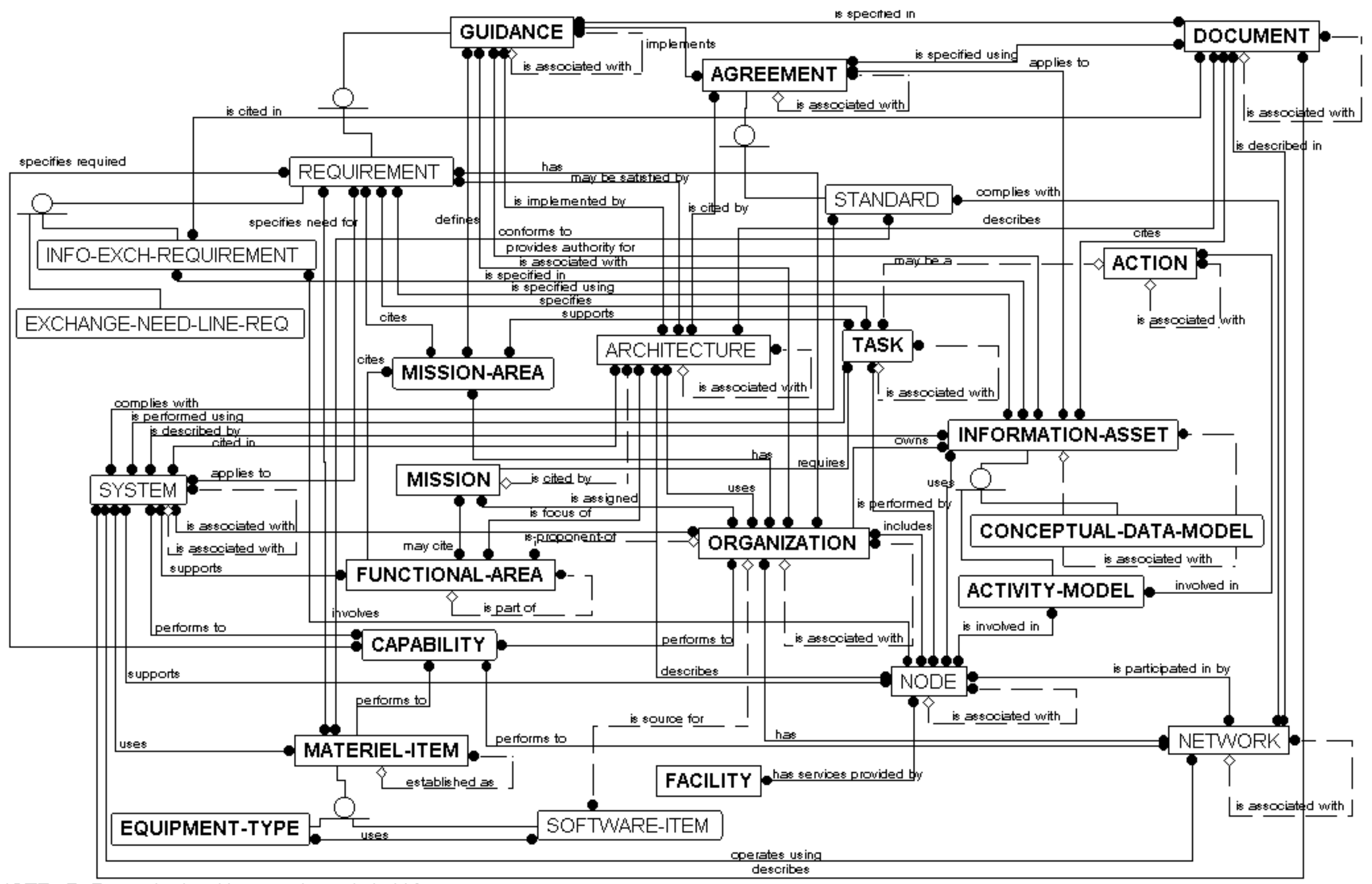


Figure A-1 . Proposed Architecture Data Model

THIS PAGE LEFT INTENTIONALLY BLANK

APPENDIX B

The following figure shows a high-level overview of the C4ISR Core Architecture Data Model:



NOTE: DoD standard entities are shown in bold font.

Figure B-1: C4ISR Core Architecture Data Model

As discussed previously, efforts to establish a common data framework are key to establishing a dynamic enterprise architecture. Depending on the size and nature of the enterprise, it is likely that pieces of the architecture will be worked in relative anonymity to the other pieces; and so, establishing a common data model is vital to future efforts to integrate those pieces into a single architecture.

In constructing the proposed methodology, careful steps were taken to ensure the new data model was compliant with the CADM. Current DoD efforts are already adhering to the guidelines established in the CADM, and it is important that any new methodologies adhere to these same basic principles, to promote reuse of previous architectural efforts. The following table shows the mapping of the proposed architectural data model elements to the C4ISR Core Architecture Data Model elements.

Proposed Data Model Element	Corresponding CADM Element
Role	Node
Activity	Process-Activity/Task[4]
Information Exchange Requirement	Exchange Need-Line-IER[5]
System	System
System Information Exchange Capability	N/A
System Activity	System-Process-Activity
System Information Exchange Requirement	Information-Exchange-Matrix-Element[6]

Table B-1 – CADM Data Model Comparison

[4] The CADM confuses the activity modeling aspects of architectures by establishing two types of "operationally-oriented" activities: Process-Activity and Task. While it claims that various instances of Process-Activity in an activity-model are related by specifying information flows between pairs of the Process-Activities, it establishes that an Information Exchange Need-Line-IER goes between two tasks. This suggests that in order to establish an operational connection between two nodes, one must define the tasks and the need to exchange information between them; and then separately model these information flows using the Process-Activity-Model. This doubles the work of the enterprise architect. The proposed methodology simplifies this construct by suggesting tasks may also be modeled using the IDEF0 processes; and, therefore, there remains no need to maintain process-activity as a separate entity. As it is, the proposed "Activity" set can be modeled using either data element.

[5] Exchange Need-Line-IER is the association between an Exchange Need-Line-Requirement (a need for a physical connection between two nodes) and an Information Exchange Requirement (a logical need for information flow between two nodes) and represents the joining of the information requirement with the physical connection requirement. In the proposed data model, we simplify this concept by assuming that in all cases where there is a logical node for information to flow, there exists a physical need, thus eliminating the need to further define these two concepts separately from their joining.

[6] This relationship is overcomplicated in the CADM. The Information-Exchange-Matrix requires the architect to associate the original Exchange-Need-Line, the IER, the System, the Process-Activity, the Task, and the System-Process-Activity to make the connection between a System and an Exchange-Need-Line-IER. This relationship is greatly simplified in the proposed methodology by associating two System-Process-Activities (System Activity) with an Exchange-Need-Line IER (IER,) which already bring with them the other information elements.

As demonstrated, the data model for the proposed methodology is generally CADM-compliant, and in many ways greatly simplifies the existing model by reuse of architectural elements. The primary reason for the vast simplification is that the proposed methodology focuses on enterprise architectures in how they can be used to capture interoperability requirements. There are a multitude of other valid purposes for architectures which are not to be understated, but do not apply to the issue of interoperability. The CADM aims to provide a data model which applies to all DoD architecture efforts, and, therefore, contains many elements not applicable to this work.

The proposed methodology introduces an additional concept beyond the CADM—the System Information Exchange Capability. Although the CADM supports the concept of a Capability, and even a System-Capability, this object cannot easily be linked to an Information Exchange Requirement, Exchange-Need-Line or Exchange-Need-Line-IER. The concept of differentiating between an information exchange requirement and information exchange capability is fundamental for the proposed architectural process to be used by system and software engineers. The ability to view current information exchange capabilities versus future operational information exchange requirements is an absolutely vital element of using the proposed methodology to levy future information exchange requirements on existing and emerging systems. The current CADM construct does allow for this kind of analysis (i.e. it provides for all the necessary data elements to make this kind of assessment); but it does not bring them together at any fused location, either by matrix or data element.

THIS PAGE LEFT INTENTIONALLY BLANK

APPENDIX C

The emergence of AP-233 as an ISO standard for systems engineering data exchange is an opportunity to show the proposed methodologies are in line with current industry trends. The figure below shows the overall structure for AP-233. All the figures found in this Appendix were found in "The technical data coverage of the emerging AP-233 STEP Standard and its use in virtual enterprises," by Julian Johnson, Erik Herzog, and Michael Giblin, three of the founders of AP-233.

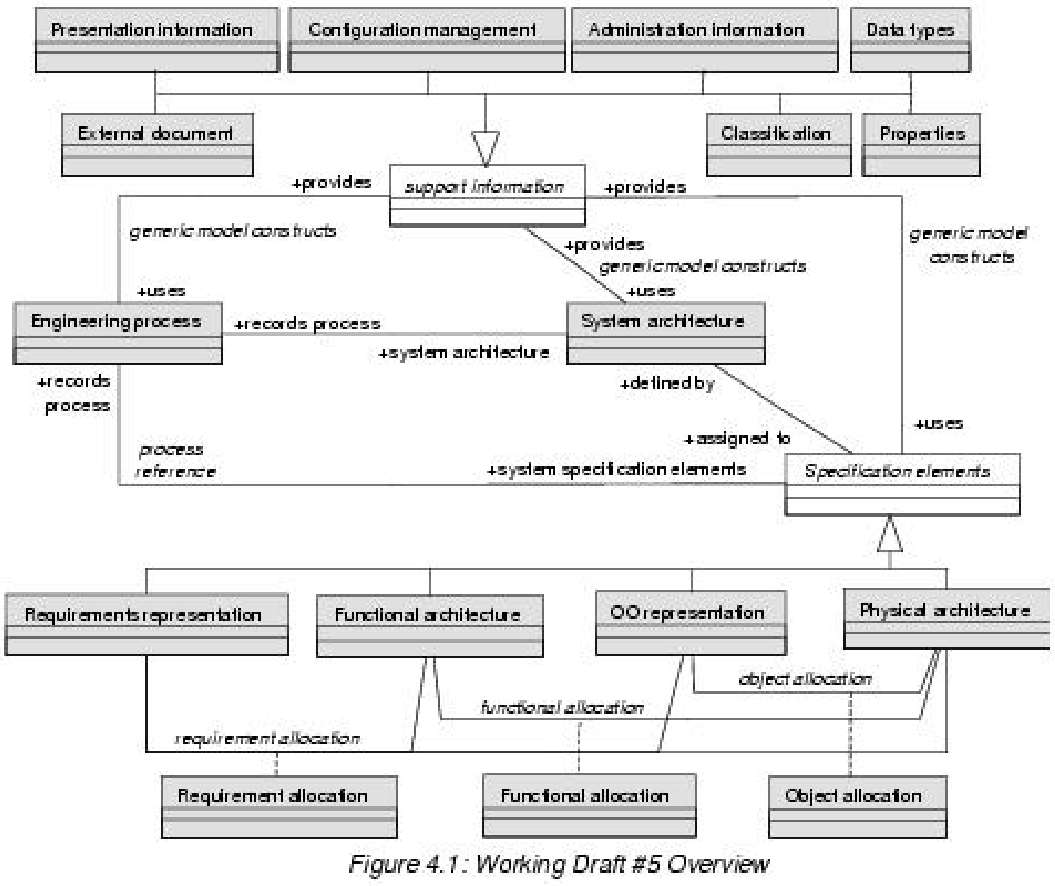
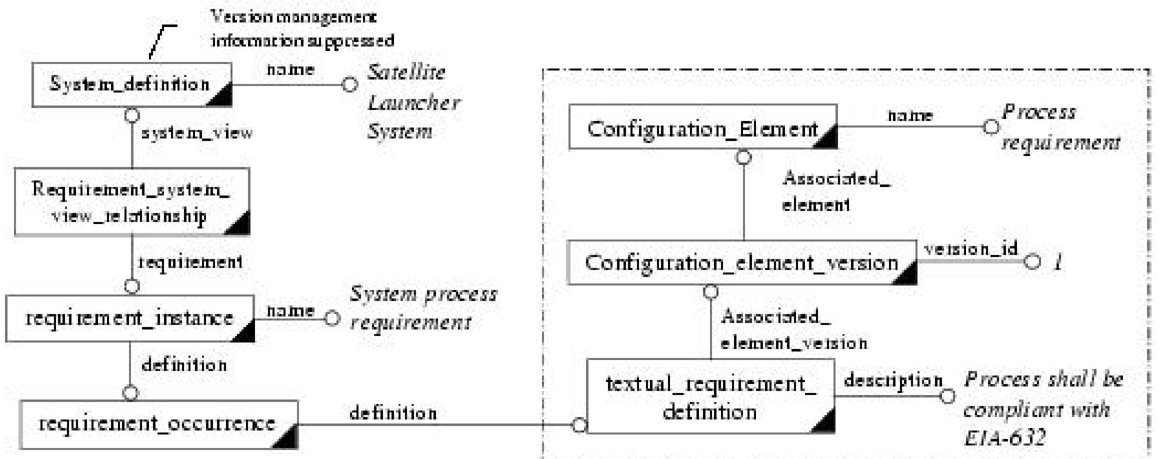


Figure C-1. AP-233 Data Model Overview [17]

Because AP-233 is still in draft, much detail on exact specifications is not available. Additionally, a complete comparison of the two data models could not be accomplished due its 'working' status. However, the overview shows many similarities to the proposed architecture data model.

First, it recognizes the need for an Object-Oriented Representation, both within the data model itself, and also as the data elements are implemented within the many tools of the Systems Engineering community. Second, AP-233 accounts for the fundamental differences between requirements and capabilities, in that it implements both a *functional* architecture and a *physical* architecture. Furthermore, it allows both *functions* and *requirements* to be mapped to the *physical architecture*, thus providing associations similar to the proposed *system function* and *system requirement*. Further detail on some of these relationships is available.

Figure C-2. Allocating Requirements to Systems [17]



Version management information suppressed
System_definition
name
Satellite Launcher System
system_view
Requirement_system_view_relationship
requirement
requirement_instance
name
System process requirement
definition
requirement_occurrence
definition
Configuration_Element
name
Process requirement
Associated_element
Configuration_element_version
version_id
1
Associated_element_version
textual_requirement_definition
description
Process shall be compliant with EIA-632

Figure C-2 shows the relationship between systems and requirements, showing the AP-233 also supports the concept of associating systems to requirements as a separate entity they refer to as a *Requirement_system_view_relationship*, but is similar in construct to the proposed *system requirement*. That is, it is an association between an instance of a *requirement* and an instance of a *system*. Furthermore, AP-233 allows for systems to be decomposed into their components systems, as seen in Figure C-3.

Figure C-4, which focuses on activities, shows many similarities to the proposed architectural data model. First, it begins the activity-modeling example with the concept of a *Work_order*, which appears to be similar to a functional area—a collection of

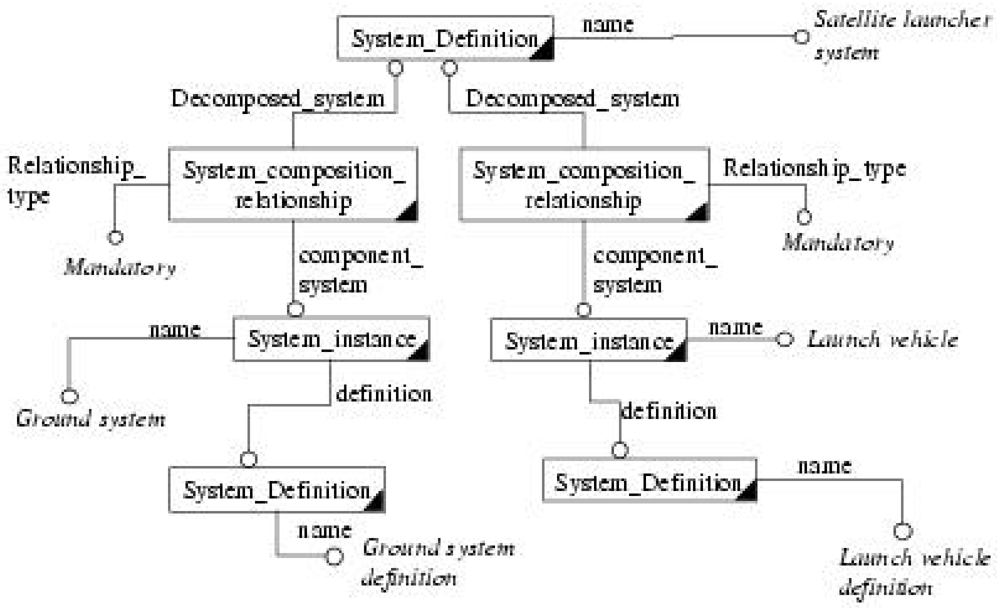
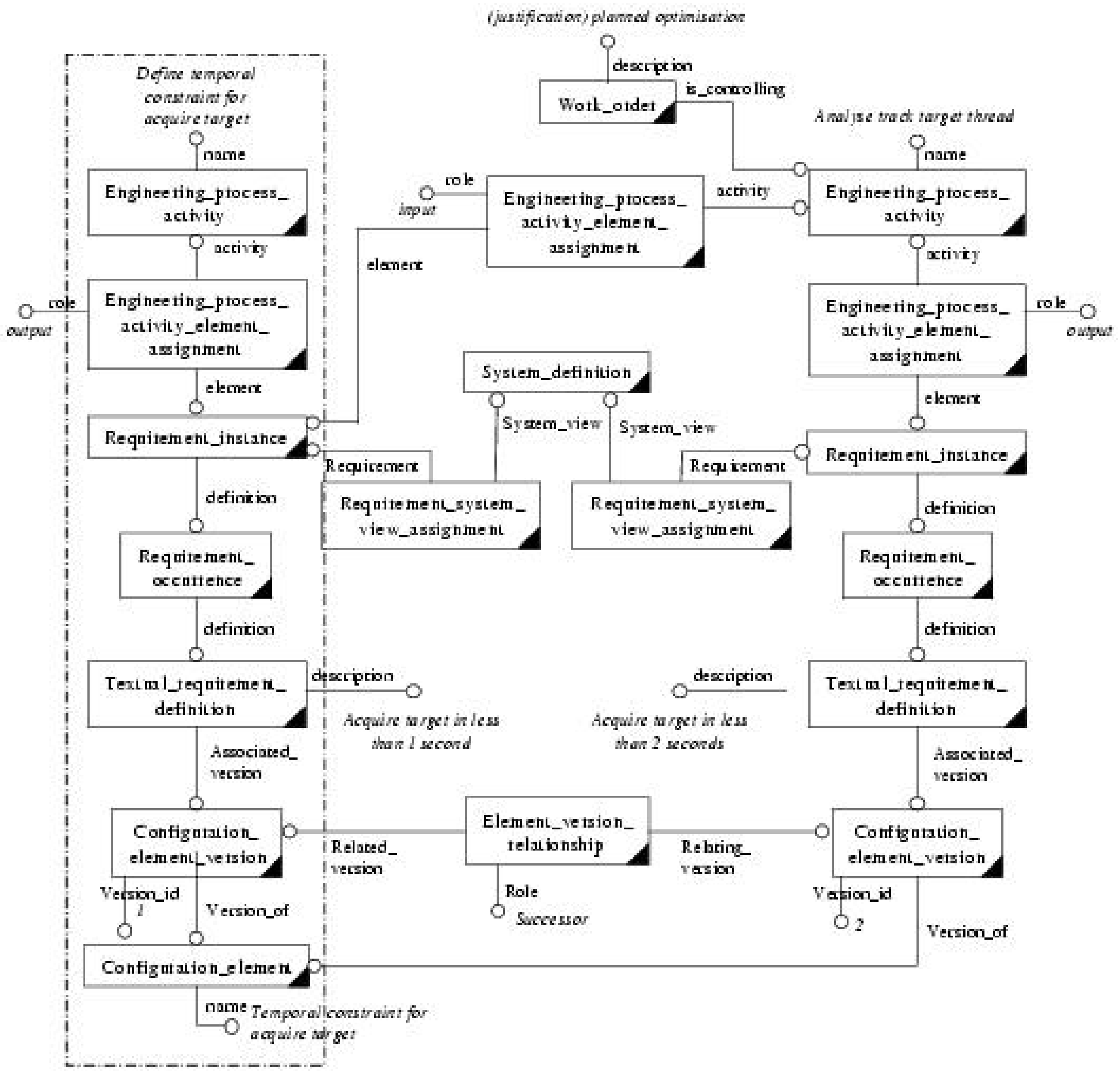


Figure C-3. System Decomposition [17]

activities that together complete a task. Second, it supports the concept of roles—groups of activities that come together to achieve a common purpose. Here seen in the block *Engineering_process_activity_element_ assignment*, where activities (and NOT requirements) are assigned to roles. Third, it recognizes that requirements are instantiations of activities as assigned to systems. This is one of the basic tenets of the proposed architectural model, and is supported in the AP-233 draft. Finally, as discussed in the previous paragraph, these requirements are assigned to systems.

There are clearly some aspects of AP-233 that are not fully accounted for. One striking difference between the proposed model and AP-233 is that AP-233 does not appear to support the concept of activity decomposition. However, it is unclear due to its draft state if this is an unsupported concept, or merely one that has not been fully architected yet.

Regardless, there are sufficient similarities between the two efforts to show that both have taken a similar approach to data modeling. It is possible that future efforts in enterprise architecting could use the AP-233 as its data standard.



(justification) planned optimisation
description
Work_order
is_controlling
Define temporal constraint for acquire target
name
Engineering_process_activity
activity
Engineering_process_activity_element_assignment
role
output
element
Requirement_instance
definition
Requirement_occurrence
definition
Textual_requirement_definition
description
Acquire target in less than 1 second
Associated_version
Configuration_element_version
Version_id
1
Version_of
Configuration_element
name
Temporal constraint for acquire target
role
input
Engineering_process_activity_element_assignment
element
activity
System_definition
System_view
System_view
Requirement
Requirement_system_view_assignment
Requirement
Requirement_system_view_assignment
Analyse track target thread
name
Engineering_process_activity
activity
Engineering_process_activity_element_assignment
role
output
element
Requirement_instance
definition
Requirement_occurrence
definition
Textual_requirement_definition
description
Acquire target in less than 2 seconds
Associated_version
Configuration_element_version
Version_id
2
Version_of
Element_version_relationship
Related_version
Relating_version
Role
Successor

Figure C-4. Activity Modeling Example [17]

LIST OF REFERENCES

[1] DoD Architecture Working Group, *C4ISR Architecture Framework Version 2.0.* 18 December 1997.

[2] Hamilton, John A., Jr., Murtagh, Jeanne L., Deal, John C., "A Basis for Joint Interoperability," Proceedings of the 1999 Command & Control Research & Technology Symposium, US Naval War College, 29 June – 1 July 1999.

[3] Hamilton, John A., Jr., Murtagh, Jeanne L., "Enabling Interoperability via Software Architecture," available online at: www.drew-hamilton.com, May 2001.

[4] Sims, D., "What is Enterprise Architecture?," available online at: www.eacommunity.com/articles/, June 2001.

[5] Spewak, S., *Enterprise Architecture Planning, Developing a Blueprint for Data, Applications and Technology,* Steven Hill, Wiley & Sons, Inc, 1992, p. xxii.

[6] Catania, Glen A., Hamilton, John A., Jr., Rosen, J. David, Melear, John, "Bilateral Interoperability through Enterprise Architecture," Fifth International C2 Research and Technology Symposium, Canberra, Australia, 24-26 October 2000.

[7] Rosen, J. David, Warwick, Jennifer L.P., Smith, Stephen A., "Architecture for Interoperability: Putting the Horse Before the Cart," Software Technology Conference 2001.

[8] Booch, Grady, Rumbaugh, James, Jacobson, Ivar, *The Unified Modeling Language User Guide,* Addison-Wesley, 1999.

[9] OASD(C3I), *C4ISR Core Architecture Data Model (CADM) Version 2.0,* 01 December 1998.

[10] Chairman of the Joint Chiefs of Staff Instruction (CJCSI) 3170.01b, *Requirements Generation Process,* 15 April 2001.

[11] Chairman of the Joint Chiefs of Staff Instruction (CJCSI) 6212.01b, *Interoperability and Supportability of National Security Systems, and Information Technology Systems,* 8 May 2000.

[12] Herzog, Erik, Törne, Anders, "Information Modelling for System Specification Representation and Data Exchange," Proceedings of the 8th IEEE International Conference and Workshop on the Engineering of Computer-based Systems, pages 136-143. IEEE Computer Society, 2001.

[13] Heimannsfeld, Klaus, Dusing, Carsten, Herzog, Erik; Johnson, Julian, "Beyond tool exchanges - Current Status and Future Implications of the Emerging ISO Standard AP-233," Presented at European Council on System Engineering (EuCOSE) - 13 September 2000.

[14] Larman, Craig, "Applying UML and Patterns, An Introduction to Object-Oriented Analysis and Design," Prentice Hall, 1998.

[15] Department of Defense Directive (DoDD) 5000.1, *The Defense Acquisition System*, January 4, 2001.

[16] Department of Defense Directive (DoDD) 5000.2, *Operation of the Defense Acquisition System,* January 4, 2001.

[17] Johnson, Julian, Herzog, Erik, Giblin, Michael, "The technical data coverage of the emerging AP-233 STEP Standard and its use in virtual enterprises," PDT Europe 2001, 24-26 April 2001, Brussels.

INITIAL DISTRIBUTION LIST