4.13 OBJECT-ORIENTED ENTERPRISE ARCHITECTURES USING INDUSTRY-STANDARD METHODOLOGY

Bruce R. Carter, CCM *
Rob Byrd, Chief Operational Architect
SI International Inc., Colorado Springs, Colorado
Technology Partner with IBM Rational

1. INTRODUCTION

The National Oceanic and Atmospheric Administration (NOAA) has undertaken, domain wide, development of enterprise architectures for accomplishing the various objectives and strategies relating to the seven strategic mission goals identified in the NOAA Strategic Plan (NOAA 2003). Architectures provide different ways to examine the mission and execution of any business or enterprise. They can describe organizational processes for reengineering purposes or for providing new technology and training. Architectures also can model doctrinal and policy implications as well as assist in the definition of the operational and system technical requirements that allow accomplishment of operational missions in an effective and efficient manner.

Vice Admiral Conrad Lautenbacher, U.S. Navy (Retired), Undersecretary of Commerce for Oceans and Atmosphere, and NOAA Administrator, has expressed it this way (David 2002)

> "*We have to build this global architecture for observing the Earth. Part of that is to build the data protocols and formats…a way to move data around that will allow maximum use by the scientific community. That's one of my pet rocks, so to speak,…."*

As presented by Rob Mairs, the National Environmental Satellite, Data and Information Service's Chief Information Officer, in a July 30, 2002 NOAA Observation System Architecture Team Overview, an enterprise's architecture can be defined in different ways. The Institute of Electrical and Electronics Engineers defines "architecture" as "*the structure of components, their relationships, and the principles and guidelines governing their design and evolution over time."* Mr. Mairs also noted that the Department of Defense (DoD), in the Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR) Architecture Framework (CAF), defines an architecture thusly (CAF 1997):

> *"An architecture description is a representation, as of a current or future point in time, of a defined "domain" in terms of its component parts, what those parts do, how the parts relate to each other, and the rules and constraints under which the parts function."*

The current DoD Architecture Framework has evolved from the CAF and formally states the Department's vision for military architectures. However, the benefits of these forward-thinking policies have not yet been fully realized in the DoD. Nearly all DoD architectures are based on the structured analysis (functional decomposition) methodology. Over the last few decades, mainly in response to the exigencies of Cold War planning and operations, significant investments have been made using this approach to create static architectures of "as is" and "to be" command and control environments, with little new insight attained. The resulting operational views (i.e., what are supposed to be the war fighter's perspective on the enterprise) tend to be foreign to the operational community, and the artifacts contribute little towards gaining new perspective or understanding regarding the value of nonmateriel solutions (e.g., doctrine, tactics, or organizational changes). Thus, many DoD operational architectures have yet to prove especially useful in supporting the development of new operational concepts or the rapid fielding (or adaptation) of new technology needed for enterprise integration.

2. A NEW APPROACH

A proven object-oriented (OO), Unified Modeling Language (UML)-based architecture and requirements management method has been developed by *SI International*, initially for DoD operational architectures, that uses industry-best practices coupled with leading-edge, web-based hosting and configuration management tools. Our method includes a collaborative, distributed environment where operational concepts can be integrated with mission requirements under a clearly defined structure that enforces horizontal integration, architecture synchronization and complete traceability to NOAA doctrine. Our method complies with all Clinger-Cohen Act mandates, wherein lie the fundamental requirements for development of architectures by federal enterprises. The result is a decision support system that helps operators better understand and communicate requirements, providing synchronization across the NOAA enterprise and enabling effective change management using industry best practices.

* *Corresponding author address:* Bruce R. Carter, SI International Engineering Inc., 1631 S. Murray Blvd., Colorado Springs, CO 80916-4552;
e-mail: bruce.carter@si-intl.com

2.1 *Understanding Requirements*

A successful requirements management process is one that involves operators and developers in an iterative and collaborative process. Because it is an object-oriented method, the *SI International* approach allows complex problems to be broken down into understandable, 'plugable' and manageable parts. This brings to mind the old adage, "How do you eat an elephant?...one bite at a time." The use of UML ensures the unambiguous articulation of desired capability (behaviors, features, and functions) in a manner that facilitates complete understanding by all stakeholders. The result is an operational architecture that can be readily reviewed, changed and evolved over time. The process allows new technology to influence operational processes such as doctrine, tactics and organizational change. Using an OO methodology, operational redundancy is easier to identify and deal with. Developmental priorities and resource allocation can be easily related to system capability and vetted by the operational community.

2.2 *Architecture Synchronization*

Object-oriented architectures are "plug and play," capability-based modules allowing reusable and adaptable core objects. This feature, when properly implemented and administered, ensures synchronization of architectures across the enterprise and also places great emphasis and dependency on interfaces, helping to move the focus from platforms to networks. Our architectures provide comprehensive traceability from required system capability to doctrine, mission requirements and external interfaces. An exclusive feature of the *SI International* approach is the ability to provide hyperlinks (via a Hypertext Markup Language [HTML] version of the UML architecture) directly to any HTML-published requirements document. Another software tool, when used as a requirements database, enables linkage between requirements documentation and specific use cases* in the UML architecture.

2.3 *Enterprise Tools Integration*

A primary problem encountered when selecting "best of breed" applications from various vendors for implementing enterprise solutions can be application integration. For example, before a new release of one application can be used, other applications must be tested to ensure they are not impacted by the new release. In many situations, this can have a profound rippling effect while attempting to spirally improve enterprise processes. This places the enterprise application user in an unnecessary role of tool integrator, whereas one would much rather focus on important enterprise issues such as enterprise modeling and change management.

2.4 *Change Management*

An OO architecture approach provides an environment that accommodates change in a cost-effective manner. Object-oriented architectures support the continuous insertion of emerging technology and spiral development while facilitating interoperability. Our process allows a free design space for follow-on developers to be innovative in satisfying operational needs. An OO architecture is equally valuable as an instrument to quickly redirect ongoing development as missions, operational concepts, and technology change. Our object-oriented architectures create an engineering environment where development occurs at a lower level with smaller and more flexible components, making the entire development process more adaptive while facilitating interoperability through architecture and component reuse.

Many legacy systems will be part of any weather-support enterprise long into the future. Our architecture approach recognizes this fact and provides the blueprint or vision that accommodates integrated and synchronized project-level development to proceed coherently in support of the overall program. The key features of our synchronized architecture solution are described in Table 1 below, along with how a weather enterprise can benefit from each feature.

The contents of this paper provide exposition of a model (Figure 1) whose methodology enables an enterprise to capture mission-wide evolving needs in both comprehensive and comprehensible fashion, and supports a spiral development process (Figure 2). The product of this methodology's use is a quantitative capability to assess materiel solutions across the entire lifecycle of a development effort from vision to requirements to architecture and design to implementation and test, including explicit formal traceabilities between related artifacts. A quantitative capability also results to assess nonmateriel solutions, changes in doctrine, changes in procedures, reorganizations for eliminating unnecessary duplications in effort, redefinition of roles and responsibilities, training, personnel management, and so on. The key to an architecture-centric methodology such described here is avoiding a stovepipe-solution approach, and concentrating instead on creating a capabilities and effects-based acquisition process. It is this change in the acquisition paradigm that leads to fundamental improvement.

For any given weather-support mission area, use of UML and related tools: (1) allow for a complete traceability from vision to requirements to architecture, and additionally from architecture to solution; (2) provide

---

* Use Case (class) A description of system behavior, in terms of sequences of actions. A use case should yield an observable result of value to an actor. A use case contains all alternate flows of events related to producing the "observable result of value." More formally, a use case defines a set of use-case instances or scenarios through specification of a sequence of actions, including variants, that a system (or other entity) can perform, interacting with actors of the system.

a seamless integration from mission architecture to operational requirements to system architecture; (3) foster integration of requirements with budgeting and acquisition; and (4) enhance revolutionary program management approaches.

For a given mission area, the vision, mission and objectives are defined. The objectives are stated in terms of observable "results of value" to be obtained without regard to any system or concept of operation. Various mission-level architectures are constructed based on combinations of principal collection, processing, evaluation, dissemination and archival systems; information–technology support systems; performance requirements and operations concepts. Mission architectures have different support structures associated with them. The cells in the mission architecture are represented by UML *use cases* in a mission-level view. Because the use cases represent "reusable" results of value, the operations analyst can begin to explore the cross-mission nature of operations and capability. We call this capability, along with the required resources (i.e., facilities and tools), the decision support system where the operational Blueprints are analyzed, managed and stored. The decision support system provides a repeatable process along with tools to perform trade studies across the operational, technical, financial and programmatic dimensions of the enterprise architectures.

Table 1: Key Features and Benefits of our Synchronized Architecture Solution

Key Features of Our Approach	Enterprise Benefits
Innovative application of Object-Oriented Systems Engineering Method: ✓ Brings operator into the solution space ✓ Operators see opportunities never seen before ✓ No more "trust me" cards ✓ Binds Operational View to Systems View ✓ Indispensable management tool, not "shelf ware" ✓ Detailed, comprehensive definition of tasks to move from "as is" to "to be," facilitating prioritization and management of the development ✓ Model becomes the system — evolves to strategic tool ✓ Common "language" that can be understood by all, from executive management to systems developers	✓ Total visibility, understanding, and substantive control of systems development through doctrine and vision
State-of-the-art modeling / architecture development tools: ✓ Accommodates new technologies (especially more sophisticated code generators as they evolve) ✓ Well suited for spiral evolution ✓ Configuration management ✓ Enterprise and systems agility in a world of changing missions ✓ Outcome is a nonproprietary model format ✓ Model itself in UML and not perishable ✓ Can relate back to existing systems and previous enterprise modeling, including Integration Definition for Functional Modeling (IDEF)	✓ Provides the ability to continuously and seamlessly upgrade technology and accommodate mission changes
Seamless, integrated end-to-end solution / no disconnects: ✓ Airtight budget protection ✓ Thought leadership with sponsors/clients and interfacing programs ✓ Return on investment relationship ✓ Eliminates redundancy ✓ Leverages reuse ✓ Test plans and training plans flow out of it	✓ Maintains total traceability of all enterprise activities, including every system function, to specific customer requirements, mission requirements, or budget items
Distributed, secure, collaborative environment for developing and maintaining architectures: ✓ Allows global participation of stakeholders ✓ Ability for sponsors/clients to meaningfully understand process/system ✓ Complete security of data transfers	✓ Allows global participation of stakeholders with completely secure data transfers.

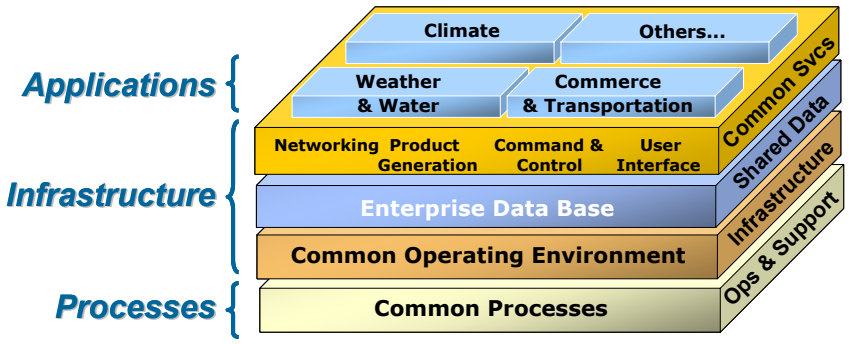

Figure 1: The Model Establishes the Enterprise Architecture's Foundation

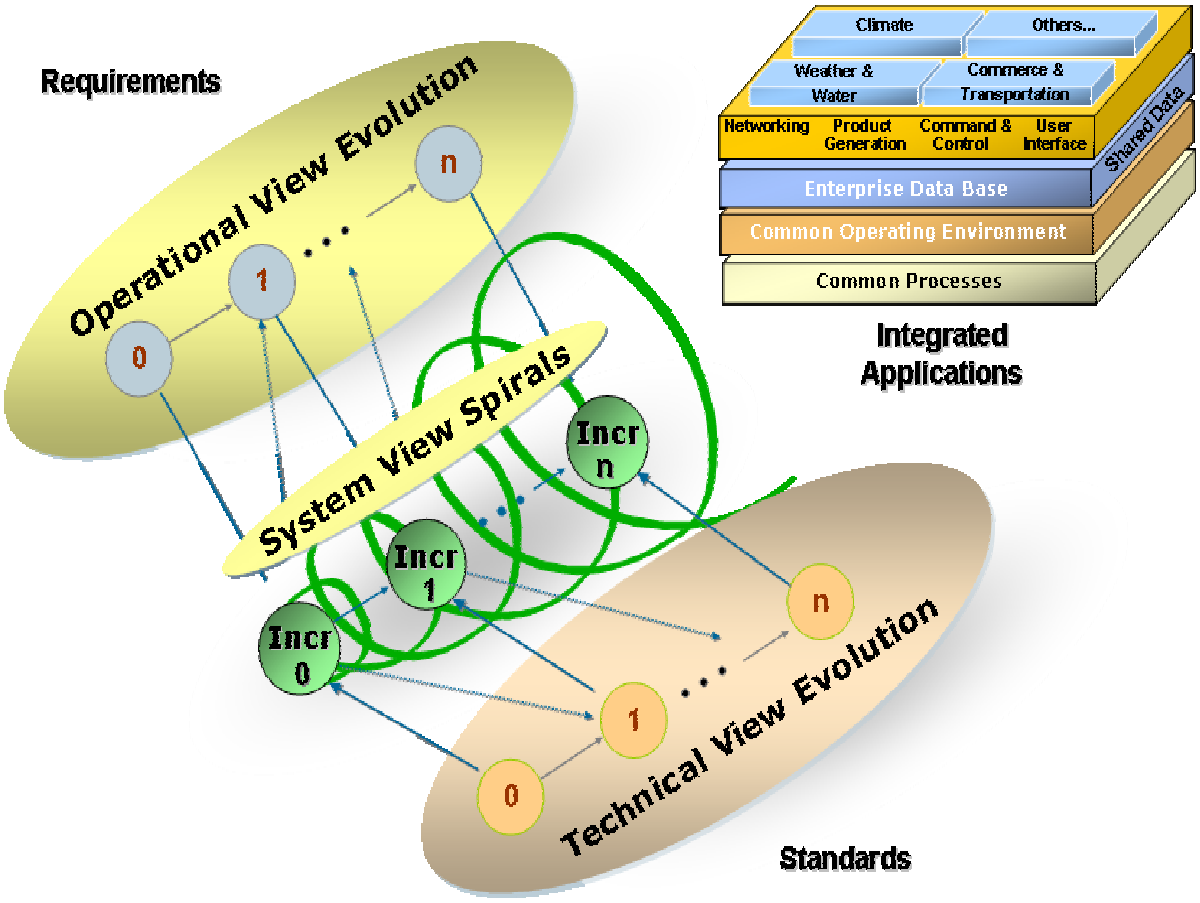

Figure 2: The Enterprise Architecture process supports and implements spiral development and evolutionary acquisition

Each element of the architecture matrix results in a mission view or relevant UML diagrams. The sum of these views or diagrams represents the system-of-systems, that is a multifunctional mission architecture. The mission architecture comprising views for systems, concepts of operations (CONOPS), requirements,

procedures, and support services is the foundation and provides guidance for operational architectures. In addition to integrating the elements of the architecture such as requirements, operations concepts and support services, the *Mission Capability Package* also links budget to elements of the architecture. This allows for "airtight" traceability from mission to funding. As the mission capability is implemented, the cost of each element can be monitored and assessed in the *context of operations*.

Our blueprint definition methodology can be described as a systems development *process*. This approach is also an *instance* of the Rational Unified Process® (RUP®), a process framework used for creating systems development processes based on successful industry philosophies and practices. The approach is also a proven methodology for developing and representing operational and enterprise architectures.

3. RATIONAL UNIFIED PROCESS

A process has four roles (Booch 1995):

- ✓ Provide guidance as to the order of a team's activities
- ✓ Specify which artifacts should be developed and when
- ✓ Direct the tasks of individual developers, and the team as a whole
- ✓ Offer criteria for monitoring and measuring a project's products and activities

The RUP® (Kruchten 1998) was developed by IBM Rational Software to address the industry-wide need for a process framework built on the collective experience of thousands of successful development efforts. From RUP®, a development organization can instantiate a process employing proven development practices. Its goal is to develop and maintain systems of high quality in a repeatable, predictable fashion. RUP-SE® is a variant of RUP® developed for system engineering applications and provides the necessary requirements management elements to describe systems and system needs. The characteristics of RUP-SE® relevant to any enterprise are discussed in the following paragraphs.

The RUP® framework is based on six fundamental best practices identified over time as those practices that are likely to improve the probability of the successful development and deployment of software systems. As depicted in Figure 3, these six best practices include:

- ✓ Develop systems iteratively
- ✓ Manage requirements
- ✓ Use component-based architectures
- ✓ Visually model software
- ✓ Continuously verify quality
- ✓ Control changes

3.1 *Develop Enterprises Iteratively*

Classic development processes follow a "waterfall" life cycle, in which the project is said to be in one of the following stages, each of which precedes the next:

- ✓ Requirements Analysis
- ✓ Analysis and Design
- ✓ Implement and Deploy
- ✓ Test and Evaluate

The waterfall approach tends to mask project risks by erecting artificial milestones leading from one stage to the next. Mistakes made eventually become apparent, but usually only after it is too late to fix them inexpensively. An alternative to the waterfall approach is the iterative and incremental process (Figure 3). In this approach, building on the work of Barry Boehm's spiral model (Boehm 1998), the identification of risks to a project is forced early in the life cycle, when it is possible to attack and react to them in a timely manner.

Figure 3: An Iterative and Incremental Process

This approach is one of continuous discovery, invention, and implementation, with iteration cycles forcing the team to drive the project's artifacts to closure in a predictable and repeatable way. The iterative approach uses a much more concrete measure of progress--executable or set of executables passing the tests assigned to it for the iteration.

Developing enterprises iteratively addresses a number of root causes of problems in enterprise development:

- ✓ Misunderstandings between various stakeholders are made known early in the project
- ✓ Early user feedback is enabled, encouraging elicitation of the actual requirements
- ✓ High risk issues are more likely to be addressed early in the project
- ✓ Since iterative development yields a testable executable at the end of each iteration, testing is enabled very early in the project
- ✓ Inconsistencies among various project artifacts are detected early
- ✓ Team workload can be managed more successfully
- ✓ Important lessons are learned early in the project so they can be leveraged to continually improve the process
- ✓ Stakeholders can be given concrete evidence of project status

3.2 *Requirements and Change Management.*

Requirements change—the larger and more complex the project becomes, the more likely it will be

that change is necessary. There are many reasons for embracing change in a system, among them include:

- ✓ The problem to be solved actually changes during development
- ✓ The customer changes their mind about what is needed, perhaps because of feedback from an iteration
- ✓ It is found that some requirements are incorrectly stated, or misunderstood -- often, again, due to feedback
- ✓ The external environment changes
- ✓ An initial delivery of the new system has been completed—it thus begins driving, however insidiously, its own requirements

The classic approach to requirements management is to resist the forces of change. The result of this action in today's complex development environments is simple: the wrong system is developed. Change in a complex system is continual; it must be accounted for from the outset of the project and managed appropriately. Active requirements management encompasses three activities:

- ✓ Eliciting, organizing and documenting the system's required functionality and constraints
- ✓ Evaluating changes to these requirements and assessing their impact
- ✓ Tracking and documenting trade-offs and decisions made about them

Requirements management is therefore much more than just writing the requirements down. This practice also offers a number of solutions to the root causes of software development problems:

- ✓ A disciplined approach is built into requirements management
- ✓ Communications are based on well-defined requirements
- ✓ Requirements can be prioritized, filtered, traced, and queried upon
- ✓ An objective assessment of functionality and performance is possible
- ✓ Inconsistencies are more easily detected
- ✓ With suitable tool support, it is possible to provide a repository for a system's requirements, the attributes of those requirements, and traceability of each requirement to others it influences

3.3 *Use Component-Based Architectures*

Each stakeholder for a project brings a different agenda to that project. We should expect stakeholders to have a different perspective on any given project. Their views also change as time passes over the project's lifetime. The enterprise architecture is the most important deliverable that can be used to cater to, present to, and manage these different viewpoints.

An enterprise architecture encompasses the set of significant decisions about these things:

- ✓ The organization of the system
- ✓ The selection of the structural elements and their interfaces by which the system is composed
- ✓ The behavior of those structural elements, as specified by the collaborations among them
- ✓ The composition of these structural and behavioral elements into progressively larger subsystems
- ✓ The architectural style that guides the development of the architecture

The decisions about these aspects of a system are made in consideration of many attributes that often conflict—not only structure and behavior but also attributes of usage, functionality, performance, resilience, reuse, comprehensibility, economic and technology constraints and trade-offs, and even aesthetic concerns.

The importance of building resilient architectures cannot be underestimated. Truly resilient architectures enable economically significant degrees of reuse, isolate hardware and software dependencies that may be subject to change, and improve maintainability. In systems as large and complex as those envisioned in conjunction with NOAA, the economics are so large that resilience in architecture very likely means the difference between success and failure. The Common Object Request Broker Architecture (CORBA) and Sun's Enterprise JavaBeans (EJB) are two of many examples that offer conclusive proof that this kind of reuse is practical and efficient, and offers reuse on a much larger scale than previously was possible in the software arena.

Coupled with the practice of developing systems iteratively, using component-based architectures involves the continual evolution of the architecture. Iterations produce an implementation architecture that can be measured, tested, and evaluated against the system's requirements. The result is a quantitative milestone that permits repeated mitigation of the most important project risks over time.

Using component-based architectures offers a number of solutions to the root causes of system development problems:

- ✓ Components facilitate resilient architectures
- ✓ Modularity enables a clear separation of concerns among elements of a system that are subject to change
- ✓ Reuse is facilitated by leveraging standardized frameworks (e.g., commercially available components)
- ✓ Components provide a natural basis for configuration management
- ✓ Visual modeling tools provide automation for component-based development

3.4 *Visually Model Systems*

A model is a simplification of reality that completely describes a system from a particular perspective. Models aid in understanding the system and permit us to better manage the system's complexity. Modeling helps the development team visualize the enterprise,

but it is also used to specify, document, and even construct the structure and behavior of a system's architecture. When a standard modeling language such as UML is used, communication becomes far less ambiguous as the team learns to use the various artifacts of the modeling language as a standard for communicating concisely and precisely.

Visual modeling tools facilitate model management by permitting various details to be hidden or revealed as the modeler wishes. The modeler exposes the architecture through the various views called for by the modeling language, and tailors these views to enhance clarity. The better modeling tools aid the development team in maintaining consistency between many of a system's artifacts—its requirements, architecture, designs, tests, and implementations.

When coupled with iterative development, visual modeling improves the development team's ability to expose, communicate, and assess architectural changes, and, again, keep them synchronized with other project artifacts. Visual modeling offers a number of solutions to the root causes of system development problems:

- ✓ Nonmodular and inflexible architectures are exposed
- ✓ Use cases and scenarios specify behavior with much less uncertainty than traditional means
- ✓ Models clearly capture system architecture and design
- ✓ Detail can be revealed or hidden as necessary
- ✓ Unambiguous designs reveal their inconsistencies more readily
- ✓ Application quality is enhanced from the start with good architecture and design
- ✓ Visual modeling tools provide extensive support for UML modeling and synchronization with requirements, implementations, and other project artifacts

3.5 *Continuously Verify System Quality*

As Figure 4 conceptually depicts, system problems are often two or three orders of magnitude more expensive to find and fix after system deployment. This underscores dramatically the need to continuously assess system quality with respect to all relevant quality factors such as functionality, reliability, performance, and usability.

Classic quality verification in system development involves constructing the system first in its entirety, then developing and exercising tests for the system. Unfortunately, some development teams have decided to employ iterative development techniques but have neglected to include testing of the system at the end of each iteration.

Continuous quality verification involves creating tests and other quality gates for the system from the beginning. System requirements and its design are used as inputs. Resulting tests are exercised in the most automated way possible, as frequently as possible, and at least as part of the conclusion of each and every iteration. Risks are reduced when the product is tested at appropriate points in each iteration. If the product is not tested, developers simply can never know definitively whether any of those risks have been corrected.

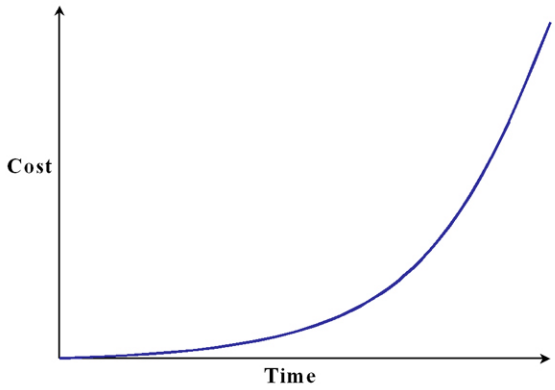

Figure 4: The Cost of Fixing System Problems Over Time

Continuous quality verification offers a number of solutions to the root causes of software development problems:

- ✓ Project status assessment is much more objective; test results, not paper documents, are evaluated
- ✓ Objective test assessments expose inconsistencies in requirements, design, and implementations
- ✓ Testing and verification are focused on areas of highest risk, thereby increasing quality and effectiveness
- ✓ Defects are identified earlier, often significantly reducing the cost of repair
- ✓ Automated testing tools provide testing for functionality, reliability, and performance

3.6 *Control Changes to the Enterprise*

Multiple developers and other contributors characterize large system development efforts. They are organized into many different teams, often at several different sites, and they are tasked to work together on multiple iterations, releases, products, and platforms. A disciplined change control process is mandatory to avoid the inevitable chaos that otherwise results.

Coordinating the activities of developers and teams and the artifacts they generate involves establishing repeatable workflows for managing changes to all project artifacts. A byproduct of this coordination is better resource allocation based on the project's priorities and risks as it actively manages the work on those changes across iterations. Coordinating iterations and releases involves establishing and releasing a tested baseline at the completion of each iteration. Maintaining traceability between the elements of each release, as well as across multiple, parallel releases, is

essential for assessing and actively managing the impact of change.

Controlling changes to the enterprise offers a number of solutions to the root causes of development problems:

- ✓ The workflow of requirements change is defined and repeatable
- ✓ Change requests facilitate clear communications
- ✓ Isolated workspaces reduce interference among team members working in parallel
- ✓ Change rate statistics provide good metrics for objectively assessing project status
- ✓ Workspaces contain all artifacts, facilitating consistency
- ✓ Change propagation is assessable and controlled
- ✓ Changes can be maintained in a robust, customizable tool or set of cooperating tools available for that purpose

In summary, RUP® strongly supports many of the principles already adopted and explained elsewhere in this paper. RUP® also is a product whose artifacts can be used to develop and document a living process for the use of every developer and other contributor involved in a system development effort. This process will be tailored to extract the most efficiency and highest quality from the development team. Such a process is essential to the success of that effort.

4.0 RUP-SE®

RUP-SE® is a variant of RUP® developed for system engineering applications with either or both of the following characteristics:

- ✓ Architecturally significant deployment issues
- ✓ Concurrent hardware and/or software development efforts

Certainly many of the architectures to be developed for 21st century enterprises will frequently have both these characteristics. However, many of the concepts in RUP®, which was designed for pure software systems, are completely applicable for the more sophisticated environments addressed by RUP-SE®.

The problem of delivering a desired system is a superset of the problem of delivering the desired software, in general. Systems engineering addresses a broader set of requirements than are normally addressed in software efforts. Even so, almost all software development efforts contain some elements of the system problem. RUP-SE addresses the larger problem of system specification, analysis, design, and development.

A major discipline addressed by RUP-SE is business modeling. The discipline is not fundamentally changed from its exposition and use in RUP, but its context is the larger system that is a superset of the software.

RUP-SE also addresses the architecture of the entire system as it evolves from the business architecture or operations architecture. As the architecture is developed, it evolves from a general, low-detail specification to a much more specific, detailed specification. We can describe these in terms of levels, such as business, analysis, design or implementation.

RUP-SE® also makes a clear distinction between allocated and derived requirements. A requirement is *allocated* if a system requirement is assigned to an architectural element. A requirement is *derived* if it is determined by studying how the architectural element collaborates with others to meet a system requirement.

The use of derived requirements for systems collaborating to carry out use cases (valuable functionality) is called *logical decomposition*. Similarly, determining subsystems by allocation is called *functional decomposition*. As before, experience has shown that functional decomposition is inferior to logical decomposition. Logical decomposition is essential for the development of quality systems.

One aspect of the systems problem is to specify a set of system use cases and supplementary requirements that, if met, would provide for a system that meets its business purpose. It follows that the system requirements are derived from an understanding of the business model. The system architectural elements in the analysis model are subsystems, localities (logical elements where fragments of system execution can occur) and processes. In the requirements analysis discipline, requirements for each type of element are determined.

There is a process pattern for deriving requirements for architectural elements:

- ✓ Determine the requirements for a given business model
- ✓ Decompose that model into elements, assigning roles and responsibilities to the elements
- ✓ Study how the elements collaborate to carry out the model requirements. This usually involves some form of collaboration specification, e.g., a UML collaboration diagram
- ✓ Synthesize the analysis of the collaboration to determine the requirements for the elements

This pattern is well known. With the business model in place, the RUP-SE® method for deriving system requirements is by partitioning the enterprise into the system and its actors (external elements that interact with the system). Then how the system and its actors collaborate to meet the business requirements is studied to determine the system requirements.

RUP-SE® applies to this pattern for deriving system requirements using processes called use-case flow-down and supplementary requirements flow-down.

Use-case flow-down begins by choosing an architecturally significant set of use cases. For each use case, a flow of events is developed. Then a description of the interactions between the system actors and the system itself is defined. The system responses are “black box;” that is, they make no reference to the architectural elements. Subsequently, initial subsystem and process models are derived using standard object-oriented analysis and design techniques.

With initial subsystems, localities, and processes defined, specifying how these elements participate in carrying out the use case revisits the flow of events. This is now a white-box view, since the flow of events now refers to design elements of the system.

Supplementary requirements flow-down is actually accomplished concurrently by specifying budgets for requirements that are allocated to individual black-box, flow-of-event steps. Later, when the flow of events has been expanded to include design elements, black-box supplementary requirements allocations are subdivided appropriately among the derived white-box steps.

Because this process can be applied recursively as needed, the RUP-SE® organizational approach to deriving design elements from use cases scales to very large programs. We take advantage of decomposition of the system into subsystems and localities with their derived requirements. Each of these analysis model elements is suitable for concurrent design and development. UML subsystems might now be assigned to separate development teams, and localities to hardware development or acquisition teams. Each team works from surveys of the recursively derived use cases to develop their portion of the design and implementation models.

A central feature of RUP® is that the system is developed in a series of iterations, each of which adds and thoroughly tests functionality. The timing and content of iterations should be captured in an Iteration Plan early in the project, and updated continually to reflect the emerging understanding of the system and changing needs.

The content of an iteration is specified by what use cases and supplementary requirements are realized by the components developed in the iteration. Each iteration is tested by the subset of applicable system test cases. Formal traceability should be developed and maintained to provide a basis for deriving iteration plans for subsequently specified subsystems and localities. A good plan will provide opportunities to identify and resolve technical risks early, once again far earlier than the typical panic of a waterfall-based integration and testing phase.

Thus, RUP-SE® provides system development teams with the advantages of RUP's best practices while providing a setting for addressing overall system issues. In particular, RUP-SE® is neither hardware nor software centric; instead, it provides for ongoing collaboration of business analysts, architects, and systems engineers to solve the system problem jointly.

5.0 ENTERPRISE ARCHITECTURES

A successful enterprise architecture initiative requires critical operational descriptions for given mission areas and their objectives; the ability to link to and evolve to a system architecture; and, for interoperability and integration reasons, the ability to realize discovery opportunities for reuse valuable to the reengineering process. Known as business modeling by industry, UML enterprise architectures capture the core of operational needs such as nodes, elements, activities, processes, organizational relationships, information exchanges and, most importantly, vision and doctrine.

In essence, an enterprise architecture captures all the essential views identified by the pertinent architecture framework. Important to *SI International*'s object-oriented approach is that the objects in the model realize reuse, essential for recognizing and planning for interoperability and integration as well as establishing the framework for bonding the operational view to the system view with total traceability. *SI International* has developed a methodology that realizes this goal.

Our modeling approach also embraces the open-system concept – modular growth, controlled change, and scalability for interoperability in the rapidly evolving environment. The open-system concept, when implementing standards-based architecture, allows for incremental technology insertion.

Our experience has shown that architecture tools require collaboration and configuration management among large teams in order to successfully architect a large enterprise. Because IBM Rational Rose® was originally developed for large-scale software engineering, the tool scales well for enterprise modeling using the UML. IBM tools such ClearCase® provide effective mechanisms for configuration control *Mission Capability Packages* at the team or individual level. IBM's ClearCase Multisite® tool allows the configuration management scheme to span multiple servers in multiple locations across the country. Data synchronization occurs at any predetermined time throughout the project. In addition, IBM's ClearQuest® provides anomaly tasking and tracking across the architectural enterprise; this ensures anomalies are tracked and corrected across the architectural teams. These tools are well integrated, allowing the architects to focus on the business of force transformation using the UML and an appropriate architecture framework rather than dealing with tool integration issues.

Each use case is value based and contributes roles, nodes, systems, activities, functions, and behaviors to successfully accomplish the desired value based result—e.g. *Measure of Combat Capability and Cost* (Figure 6). Roles, nodes and systems are depicted with UML actors (the stick figures shown in Figure 6). These techniques are only useful when sufficient understanding can be taught to less experienced architects.

Our experience has shown that object-oriented methods are repeatable and the techniques are trainable. The UML methods themselves are useful for documenting the internal processes of the architecture team (Figure 10). In essence, the tools are useful for developing internal standing operating procedures (SOPs), reducing training, and improving on consistency and quality. Using the SOPs, the architects can successfully and repeatedly develop the contents of value-based *Mission Capability Packages.*

As the "parent use case," the operational concept (e.g. *Develop Measure of Combat Capability and Cost* [Figure 6]) can be understood and developed using UML Activity Models. The Activity Diagram for *Develop*

Measure of Combat Capability and Cost (Figure 6) clearly outlines the process to achieve the value-base objective, and the reader can easily determine that the Stakeholder submits a Decision Need Request from which an Analysis Plan is developed. As shown, many of the activities presented <<include>> additional use cases with their own processes to create relevant products (objects) needed by the use-case activities. Various products (objects created by other use cases) are used by the *Create Warfare Simulation* use case to provide Low-Level Metrics necessary to determine Combat Capability and Cost. The Decision Support System (DSS) Agent delivers the findings, Combat Capability and Cost, to the requesting Stakeholder.

The information flow is clearly understood by the mission or operational analyst and is useful in system development. Each object (e.g. LowLevelMetrics; see activities 5.7 and 5.8 in Figure 6) is either produced or consumed by the activity. In addition, the objects can move between nodes and organizations in the form of information exchanges. The object movement is captured by a System Operational Sequence (SOS) depicted in Figure 7.

Information exchanges depicted by the DoD Architecture Framework's Operational View (OV)-2 (Table 2), the Node Connectivity Description (NCD), provide the information flow between nodes (both operational and physical) and roles (Figure 5). The architecture continues to mature with the NCD Sequence (Figure 8), which provides a time-order view of the activity and is automatically generated by the NCD (OV-2).

Using the sequence view, behavior between nodes and roles becomes better understood, allowing the development of the Use Case Specification (UCS) (Figure 9). The UCS, based on RUP®, is where the words are added to the views to describe the system behavior. The UCS describes the desired capabilities as if the system were a "black box" using the principles of RUP-SE®. The process is iterative and the views are updated as the UCS becomes mature and captures the essential requirements. Depending on the purpose of the enterprise model, we have experienced varying usages of this method. Some enterprises require only conceptual descriptions of the processes, while others require detailed system descriptions to serve as descriptions of requirements to build systems.

If system or developmental requirements are the objective, the enterprise architects can apply the RUP-SE® black-box technique using the UML Sequence Diagram (Figure 7). This revolutionary new breakthrough allows the architect and subject-matter expert to collaborate on the desired system behavior. A new system object (e.g. Figure 7 shows the new "DSS" object) is used to articulate system behavior requirements to the system object as if the object were the black box. Using the tool's documentation view, further refined description of the behavior is developed, eliminating any misinterpretation of the system behavioral requirement.

The discussions and views shown provide a general overview of the process; however, there's much more involved in successfully creating a use case of value for the enterprise manager and/or operator. This overview is only intended to provide the reader a general understanding of the value of the architecture-centric methodology.

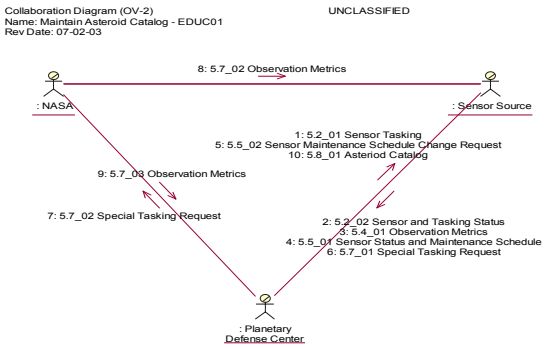

Figure 5: Node Connectivity Description (OV-2)

Although the technique can be used to manage complex requirements, some enterprises find the Activity Models and related object data flows to be the desired level of detail for enterprise decisions. Finally, enterprises can benefit greatly with a method that has uses at varying levels of abstraction, allowing further decomposition to the system requirement should the need arise as a better understanding of the enterprise is realized.

An activity model (Figure 10) can depict the high-level process used by *SI International* to build operational architectures. This process has been developed and used successfully on a variety of DoD programs. Proven repeatability, the systematic process, and numerous feedback loops ensure synchronization of any Enterprise's required Architecture Framework views as the enterprise architecture evolves over time to remain current with mission needs and priorities.

6.0 INTEGRATED REQUIREMENTS MANAGEMENT

An OO enterprise architecture facilitates the development of integrated capability. The enterprise architecture is an overarching picture of mission functions and the information exchanges required for mission accomplishment. Enterprise architectures reflect doctrinal and policy implications and become an enabling technology to remove or significantly reduce redundancy in requirements and documents while maintaining compete traceability. They assist in the definition of the physical and system requirements that allow the operator to accomplish any mission in an effective and efficient manner. The enterprise architecture is meant to be a living concept, which allows for new missions, new functions, new technology, and new alliances. It is conceived as a "plug-and-play"

Applicable View	Framework Product	Framework Product Name	General Description
All Views	AV-1	Overview and Summary Information	Scope, purpose, intended users, environment depicted, analytical findings
All Views	AV-2	Integrated Dictionary	Architecture data repository with definitions of all terms used in all products
Operational	OV-1	High-Level Operational Concept Graphic	High-level graphical/ textual description of operational concept
Operational	OV-2	Operational Node Connectivity Description	Operational nodes, connectivity and information exchange need lines between nodes
Operational	OV-3	Operational Information Exchange Matrix	Information exchanged between nodes and the relevant attributes of that exchange
Operational	OV-4	Organizational Relationships Chart	Organizational, role, or other relationships among organizations
Operational	OV-5	Operational Activity Model	Capabilities, Operational Activities, relationships among activities, inputs and outputs. Overlays can show cost, performing nodes, or other pertinent information
Operational	OV-6a	Operational Rules Model	One of the three products used to describe operational activity—identifies business rules that constrain operation
Operational	OV-6b	Operational State Transition Description	One of three products used to describe operational activity—identifies business process responses to events
Operational	OV-6c	Operational Event-Trace Description	One of three products used to describe operational activity—traces actions in a scenario or sequence of events
Operational	OV-7	Logical Data Model	Documentation of the system data requirements and structural business process rules of the Operational View.
Systems	SV-1	Systems Interface Description	Identification of systems nodes, systems, and system items and their interconnections, within and between nodes
Systems	SV-2	Systems Communications Description	Systems nodes, systems, and system items, and their related communications lay-downs
Systems	SV-3	Systems-Systems Matrix	Relationships among systems in a given architecture; can be designed to show relationships of interest, e.g., system-type interfaces, planned vs. existing interfaces, etc.
Systems	SV-4	Systems Functionality Description	Functions performed by systems and the system data flows among system functions
Systems	SV-5	Operational Activity to Systems Function Traceability Matrix	Mapping of systems back to capabilities or of system functions back to operational activities
Systems	SV-6	Systems Data Exchange Matrix	Provides details of system data elements being exchanged between systems and the attributes of that exchange
Systems	SV-7	Systems Performance Parameters Matrix	Performance characteristics of systems view elements, for the appropriate timeframe(s)
Systems	SV-8	Systems Evolution Description	Planned incremental steps toward migrating a suite of systems to a more efficient suite, or toward evolving a current system to a future implementation
Systems	SV-9	Systems Technology Forecast	Emerging technologies and software/hardware products that are expected to be available in a given set of timeframes, and that will affect future development of the architecture
Systems	SV-10a	Systems Rules Model	One of three products used to describe systems functionality—identifies constraints that are imposed on systems functionality due to some aspect of systems design or implementation
Systems	SV-10b	Systems State Transition Description	One of three products used to describe systems functionality—identifies responses of a system to events
Systems	SV-10c	Systems Event-Trace Description	One of three products used to describe systems functionality—identifies system-specific refinements of critical sequences of events described in the operational view
Systems	SV-11	Physical Schema	Physical implementation of the Logical Data Model entities, e.g., message formats, file structures, physical schema
Technical	TV-1	Technical Standards Profile	Listing of standards that apply to systems view elements in a given architecture
Technical	TV-2	Technical Standards Forecast	Description of emerging standards and potential impact on current systems view elements, within a set of timeframes

Table 2: Essential and Supporting Framework Products

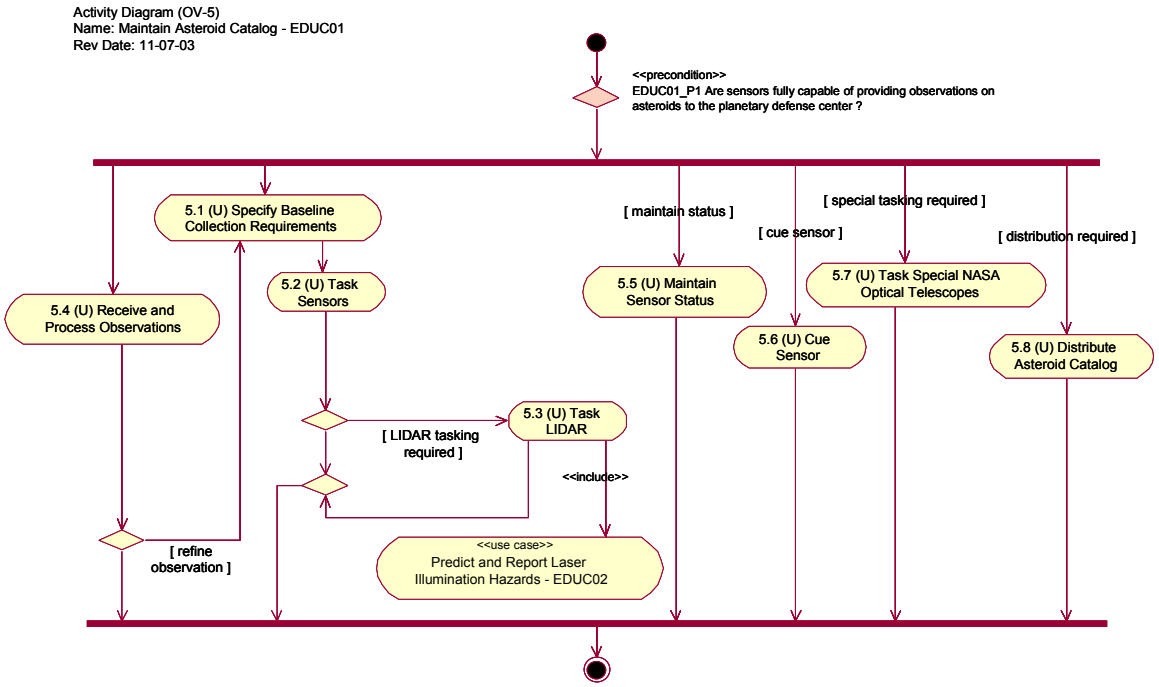

Figure 6: Activity Model (OV-5)

configuration where new mission functionalities are added to the existing system and seamlessly integrated into the operator's activities and tasks.

The enterprise architecture allows the decision maker to visualize, in pictures and words, how the organization operates and how all the required tasks are performed. Another purpose of the enterprise architecture is to provide the systems engineer with the information needed to define the systems to be developed without placing any technological restrictions on the system's design.

The nature of today's programming and software development environment is OO. By organizing operational system characteristics, processes, and activities in an OO form, software developers are no longer required to reengineer the coding framework. Core or common elements are clearly recognizable. Using this UML methodology, an enterprise architecture team is able to bind vision, doctrine, and operational processes to an enterprise's system designs.

For the NOAA, a stated key aspect of overall strategy is to "…improve…policy, programmatic, and managerial foundations…and build a corporate NOAA that facilitates the effective, timely delivery of our products and services." (NOAA 2003) Applications of the enterprise architecture methodology summarized in this paper can contribute very meaningfully to the administration's achievement of that strategy.

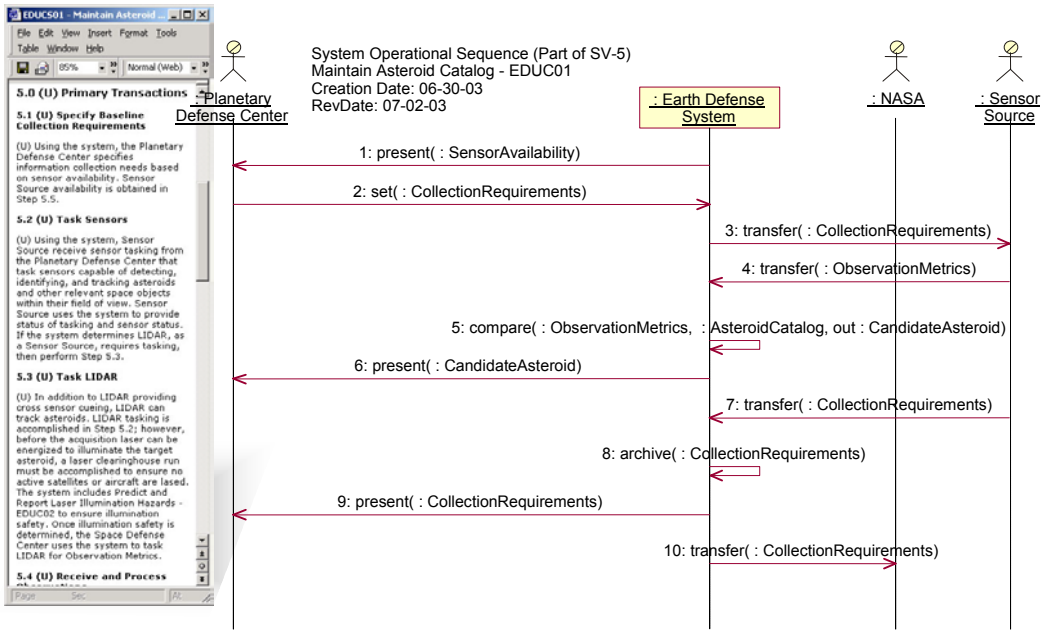

Figure 7: System Operational Sequence (Operational Portion of SV-5)

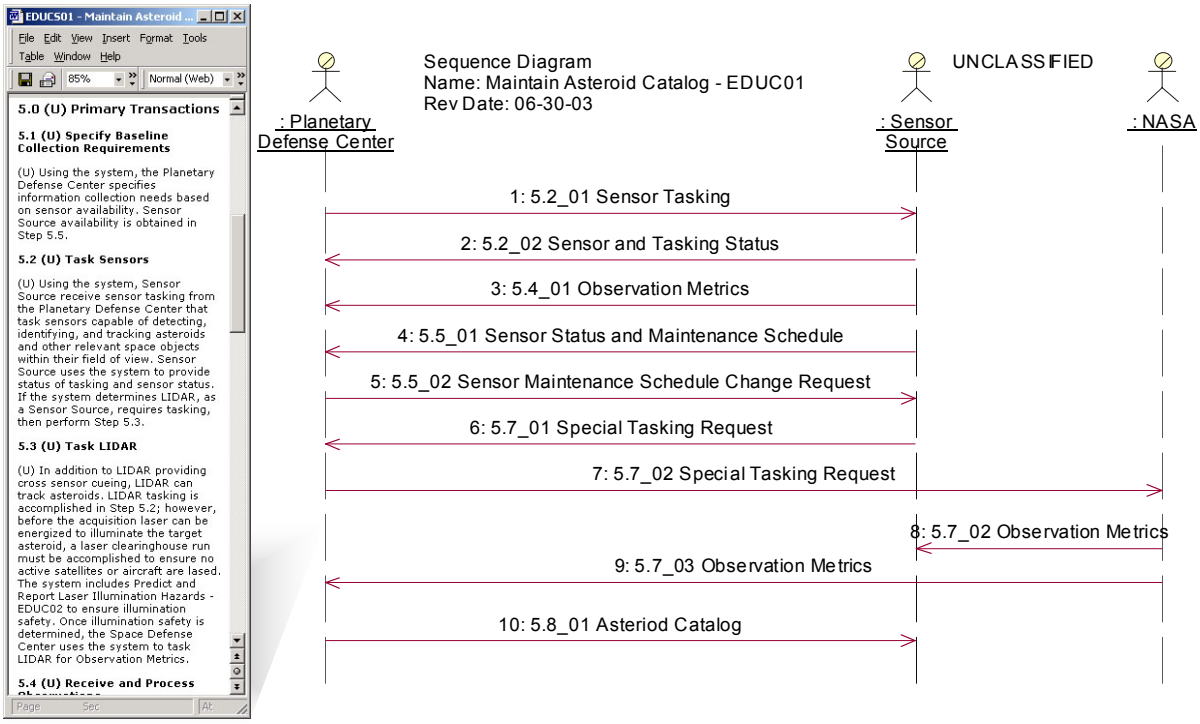

Figure 8: Node Connectivity Description (NCD) Sequence (OV-5/OV-3/OV-2)

(U) EDUCS01 - Maintain Asteroid Catalog
OPR: SI International
Phone #: (719) 235-4400
Rev Date: 11-07-03

1.0 (U) Summary
(U) Specifies activities, relationships and organizational roles required to develop and maintain an asteroid catalog sufficient to support the earth defense from foreign bodies or threats.

2.0 (U) Scope
(U) Manage sensor availability and task asteroid sensors to maintain currency of asteroid positional data and characteristics.

3.0 (U) Actors
Planetary Defense Center, LIDAR, NASA, Optical Telescopes, Universities and Labs

4.0 (U) Preconditions
(U) Sensors must be fully capable of observing the heavens to identify, track, and provide observations on asteroids to the planetary defense center.

5.0 (U) Primary Transactions

5.1 (U) Specify Baseline Collection Requirements
(U) Using the system, the Planetary Defense Center specifies information collection needs based on sensor availability. Sensor Source availability is obtained in Step 5.5.

5.2 (U) Task Sensors
(U) Using the system, Sensor Source receive sensor tasking from the Planetary Defense Center that task sensors capable of detecting, identifying, and tracking asteroids and other relevant space objects within their field of view. Sensor Source uses the system to provide status of tasking and sensor status. If the system determines LIDAR, as a Sensor Source, requires tasking, then perform Step 5.3.

5.3 (U) Task LIDAR
(U) In addition to LIDAR providing cross sensor cueing, LIDAR can track asteroids. LIDAR tasking is accomplished in Step 5.2; however, before the acquisition laser can be energized to illuminate the target asteroid, a laser clearinghouse run must be accomplished to ensure no active satellites or aircraft are lased. The system includes Predict and Report Laser Illumination Hazards - EDUC02 to ensure illumination safety. Once illumination safety is determined, the Space Defense Center uses the system to task LIDAR for Observation Metrics.

5.4 (U) Receive and Process Observations
(U) The system provides Observation Metrics from Sensor Source at the Planetary Defense Center for processing. The Planetary Defense Center uses the system to process the observations by comparing historical asteroid data. If the system correlates the observation to a known asteroid, the positional vector of the asteroid is updated. If the observations compare to no known asteroids, then a candidate asteroid is developed, and additional information regarding its position and composition is sought by further refined collection requirements and sensor tasking in Steps 5.1 and 5.2.

5.5 (U) Maintain Sensor Status
(U) The system provides ongoing status of Sensor Source sensors to include planned downtime and failure degradation notification to the Planetary Defense Center. The system aids in providing scheduled downtime coordinated with the Planetary Defense Center.

5.6 (U) Cue Sensor
(U) Since asteroids traverse the celestial sky, they may pass from a sensor's field of view. When this occurs, the system uses an LIDAR device to cue another sensor to establish track of the asteroid and continue collecting observations. Although this is an automated process, the LIDAR manager notes the cueing transaction from one sensor to another.

5.7 (U) Task Special NASA Optical Telescopes
(U) Because NASA owns and operates special optical telescopes outside the scope and purview of the Planetary Defense Center, NASA may use the system to provide specific asteroid tasking to compliment Space Research Centers sensors tasking. Information from this tasking is fed by the system to Space Research Centers for research, as well as to the Planetary Defense Center for asteroid catalog maintenance.

5.8 (U) Distribute Asteroid Catalog
(U) Periodically, or as needed, the Planetary Defense Center distributes the updated Asteroid Catalog to LIDAR, NASA, Optical Telescopes, Space Research Centers and interested users.

6.0 (U) Post-Conditions
(U) Asteroid Catalog is updated based on recent observations from the Planetary Defense System.

7.0 (U) Alternate Transactions
(U) None

8.0 (U) User Interface
(U) TBD

9.0 (U) Participating Objects
(U) TBD

10.0 (U) References
(U) NASA Publication 10-40, Asteroid Catalog Maintenance, July 2000

11.0 (U) Notes
(U) Information included in the Asteroid Catalog:
a. (U) Asteroid Orbital element set data
b. (U) Element set observation age (regency of observed position)
c. (U) Asteroid composition (material and density)
d. (U) Point of closest approach to Earth (time and location)
e. (U) Probability of impact at point of closest approach to the Earth
f. (U) Any other items of interest or significance

Figure 9: Use Case Specification

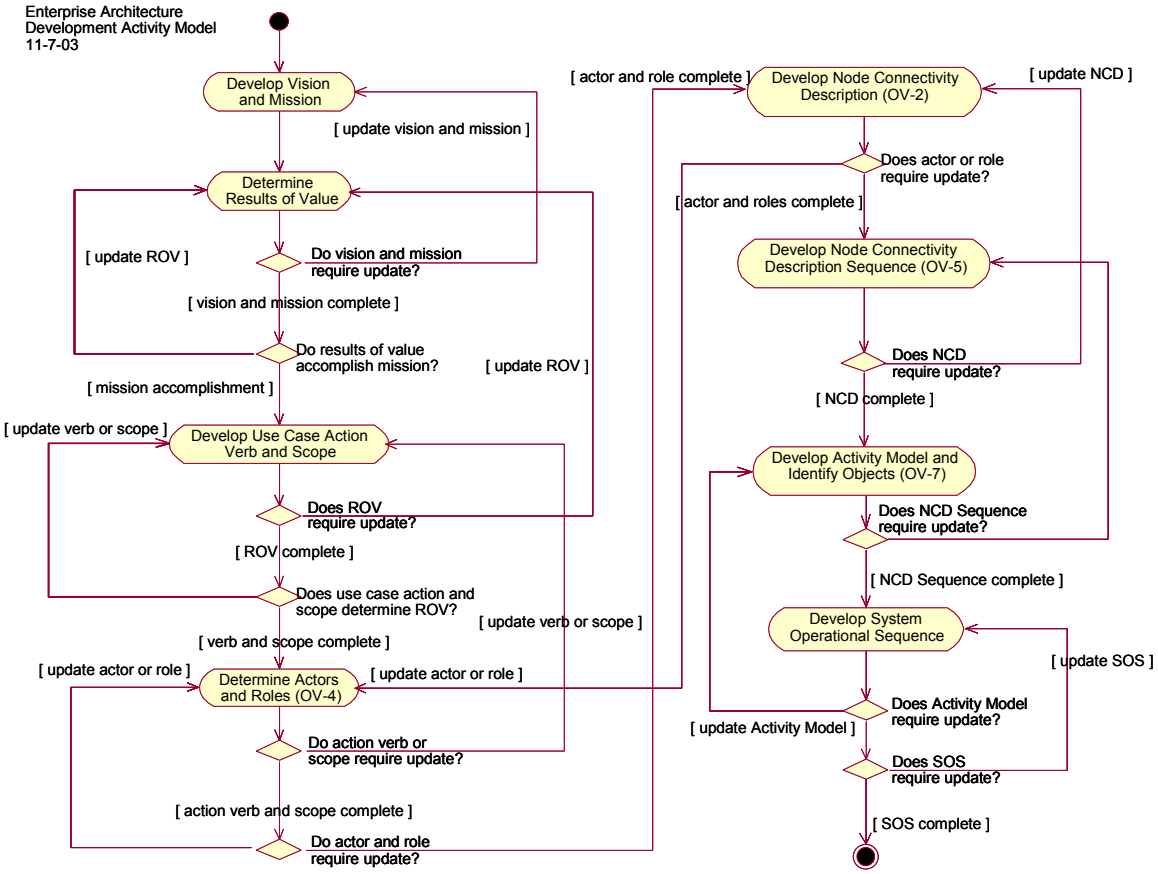

Figure 10: A mature enterprise architecture development process ensures synchronization of required Enterprise Architecture Framework views

REFERENCES

Boehm, B.W., 1988: A Spiral Model of Software Development and Enhancement. *IEEE Computer*, May, pp. 61-72.

Booch, G., 1995: *Object Solutions—Managing the Object Oriented Process.* Addison-Wesley, 323 pp.

Command, Control, Communications, Computers, Intelligence, Surveillance and Reconnaissance (C4ISR) Architecture Framework (CAF), 1997, Version 2.0.

David, L., 2002: NOAA's Big Picture Look at Earth. *Space News*, 27 November 2002.

Kruchten, P., 1998: *The Rational Unified Process, An Introduction*. Addison-Wesley, 320 pp.

NOAA, 2003: *NOAA's FY 2003 — FY 2008 Strategic Plan*, 31 March 2003. http://www.osp.noaa.gov/pdfs/FinalMarch31st.pdf