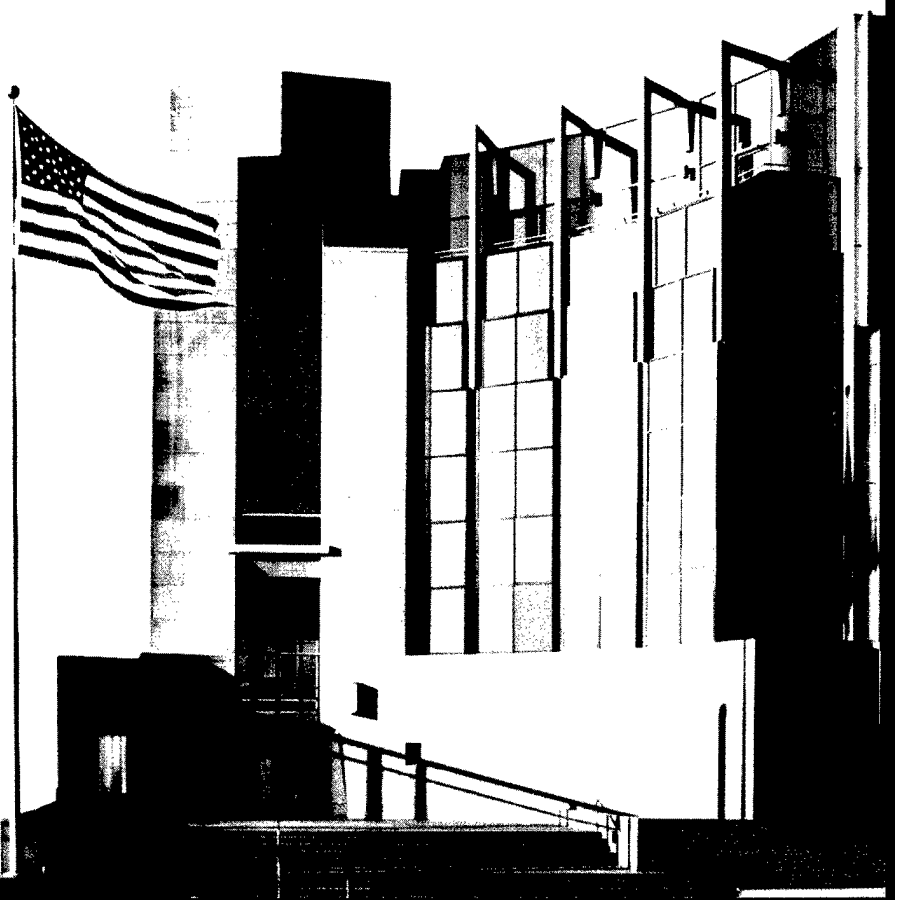
Carnegie Mellon
Software Engineering Institute

DoD Architecture Framework and Software Architecture Workshop Report

William G. Wood, *Software Engineering Institute*
Mario Barbacci, *Software Engineering Institute*
Paul Clements, *Software Engineering Institute*
Steve Palmquist, *Software Engineering Institute*
Huei-Wan Ang, *The MITRE Corporation*
Loring Bernhardt, *The MITRE Corporation*
Fatma Dandashi, *The MITRE Corporation*
David Emery, *The MITRE Corporation*
Sarah Sheard, *Software Productivity Consortium*
Lyn Uzzle, *Software Productivity Consortium*
John Weiler, *Interoperability Clearinghouse*
Art Krummenoehl, *Johns Hopkins University Applied Physics Laboratory*

March 2003

Architecture Tradeoff Analysis Initiative



Technical Note
CMU/SEI-2003-TN-006

DoD Architecture Framework and Software Architecture Workshop Report

William G. Wood, *Software Engineering Institute*
Mario Barbacci, *Software Engineering Institute*
Paul Clements, *Software Engineering Institute*
Steve Palmquist, *Software Engineering Institute*
Huei-Wan Ang, *The MITRE Corporation*
Loring Bernhardt, *The MITRE Corporation*
Fatma Dandashi, *The MITRE Corporation*
David Emery, *The MITRE Corporation*
Sarah Sheard, *Software Productivity Consortium*
Lyn Uzzle, *Software Productivity Consortium*
John Weiler, *Interoperability Clearinghouse*
Art Krummenoehl, *Johns Hopkins University Applied Physics Laboratory*

March 2003

Architecture Tradeoff Analysis Initiative

Technical Note
CMU/SEI-2003-TN-006

Contents

List of Figures

Abstract

During the Software Engineering Institute's Workshop on the Department of Defense Architecture Framework and Software Architecture, participants from government, industry, and academia discussed the similarities and differences between system and software architecture representations, and how these representations relate with one another. This technical note summarizes the activities of that workshop.

1 Introduction

The Software Engineering Institute (SEI[SM]) conducted the Workshop on the Department of Defense (DoD) Architecture Framework (DoDAF)[1] and Software Architecture on January 30, 2003, near Washington, DC. This workshop provided a forum for participants to discuss the similarities and differences between system and software architecture representations, and how these representations interrelate. The participants were invited because of their familiarity with the representations and the various approaches that apply to those representations.

This half-day workshop consisted of five presentations, which are described in the body of this report. The workshop concluded with a facilitated discussion.

This report is organized as follows. Section 2 describes the background and purpose of the workshop. Section 3 summarizes the attendees' presentations of approaches to representing software and system architectures. Section 4 describes the topics attendees discussed, and Section 5 provides a summary of the results. The appendices provide further detail about the approaches discussed.

[SM] SEI is a service mark of Carnegie Mellon University.

[1] The DoDAF is an in-progress revision of the C4ISR (Command, Control, Communications, Computer, Intelligence, Surveillance, and Reconnaissance) Architecture Framework.

2 Background and Purpose

The DoDAF is being mandated by the DoD as the basis for building representations of large-scale systems of systems. The DoDAF prescribes three major interrelated views to represent system architecture: system, operational, and technical.

While the DoDAF deals with systems of systems, there is also an entire community devoted to the design and representation of *software* architectures. For example, the SEI has developed approaches for documenting, building, and analyzing software architectures. The Unified Modeling Language (UML) has become a standard notation for describing software designs. Both UML and the SEI's "Views and Beyond" approach to architecture documentation use multiple views to represent a software architecture. However, the views most often used in the software architecture community do not correspond to the DoDAF views.

During the development life cycle, DoDAF views might serve as a basis for the development of a software architecture, but there is no accepted way of using DoDAF views as a foundation for this development. Furthermore, there is no clear correspondence between DoDAF views and notations, and those most useful for representing a software architecture. Nevertheless, because of the ubiquity of the DoDAF and a failure to distinguish between system and software architectures in some quarters, some DoD acquisition project teams attempt to fit their software architectures into the DoDAF because they believe that policy requires them to do so.

This workshop is a first step toward understanding the representational challenges involved in architecting system and software architectures, as well as trying to understand the transformation from DoDAF representations to software architecture representations.

3 Summary of Briefings

Workshop attendees described various aspects of architectural approaches in five briefings. A summary of these briefings follows, and more detail is provided in the appendices.

1. Paul Clements presented an overview of the SEI's Views and Beyond approach to documenting software architectures. Rather than prescribing a fixed set of views, as in the Rational Unified Process (RUP) or the DoDAF, this approach suggests that the stakeholders should first determine which architectural representation best captures the information they need to do their jobs. The software architects create a table showing the system stakeholders and the views that best represent their viewpoints. The architects then combine views and prioritize them until a set of views that sufficiently covers the viewtypes is selected. The architects document each view as a number of "view packets" that show, in varying degrees of detail, different elements and element relationships that would interest a stakeholder. The Views and Beyond approach suggests a template for view packets, as well as for documentation that applies to more than one view. The most important part of the latter is a mapping among views that provides a holistic picture of the total design by showing how information in one view relates to that in another.

   The Views and Beyond approach acknowledges that there is a limited number of viewtypes to represent the essential categories of views. It also recognizes that architectural styles (or known design approaches) can provide the conceptual basis for describing a software architecture.

   Appendix A contains more details of this approach.

2. Fatma Dandashi presented the DoDAF by giving an overview of the proposed changes to the C4ISR Architecture Framework. One major change is that the architectural products developed for a system architecture now depend on the life-cycle development stage and the purpose for building the architecture. For example, an architecture developed to aid budget planning requires different products from one used to assist with the development of a concept of operations (CONOPS). Likewise, an architecture developed to assist with the development of a CONOPS differs from one developed to serve as a blueprint for system construction. The details required by the architecture also change during its purpose and life-cycle phase. The DoDAF defines 22 products organized into three views; architects must select the most appropriate subset of these products to satisfy their purpose.

   Because the DoDAF is not yet published, we could not include actual text from it in this report. Instead, Appendix B contains text from the C4ISR Architecture Framework, and Appendix E describes how the DoDAF relates to the Federal Enterprise Architecture (FEA) mentioned in Item 5 below.

3. David Emery presented an overview of the relationship between the DoDAF and IEEE Std 1471-2000. This standard suggests that representations for a software-intensive system

   - provide a context that describes how a system fits into its environment
   - account for the concerns of its various stakeholders
   - fulfill the mission requirements of the system

   These requirements can be fulfilled by creating a viewpoint, which establishes the conventions by which a view is created, depicted, and analyzed. The viewpoint determines the languages that will be used to describe the view, as well as any associated modeling methods and analysis techniques that will be applied to these representations of the view. The viewpoints developed depend heavily on the stakeholders' concerns. Moreover, a view can consist of a number of architectural models or representations.

   An excerpt of IEEE Std 1471-2000 is provided in Appendix C.

4. Loring Bernhardt presented an overview of the challenges of developing architectures to evolve existing stovepiped software-intensive systems into a constellation of interoperable systems of systems. This evolution is usually done in a number of delivery blocks (e.g., every 18 months) over a number of years (e.g., 10 years). Many challenges arise because the legacy systems are often approaching technical obsolescence, and 10-year technology forecasts are unreliable. Developing architectures to evolve existing systems requires (among other things) an approach of creating and maintaining a master evolution plan (MEP) to describe how the new architecture will be developed, the impact on the "sensor-to-shooter" chain at each delivery block, and the life-cycle cost predictions. None of the standard DoDAF products capture the MEP in a satisfactory manner.

   Appendix D contains more information on analytic views within the DoDAF.

5. John Weiler presented an overview of the FEA, an architecture that is being developed for the Office of Management and Budget (OMB) to facilitate cross-agency and within-agency analysis of duplicative investments and opportunities for collaboration. Many federal agencies within the federal government have needs for systems whose features and capabilities overlap with the needs of other government agencies, and which are likely to be created from the same set of commercial software and hardware components. This overlap suggests the use of these interrelated reference models:

   - Business Reference Model (BRM)
   - Performance Reference Model (PRM)
   - Data and Information Reference Model (DIRM)
   - Application-Capability Reference Model (ACRM)
   - Technical Reference Model (TRM)

The BRM and PRM describe the objectives for the agency, and the DIRM, ACRM, and TRM describe how best to allocate resources, technology, and services to meet these objectives. To date, only the BRM is defined.

Appendix E provides an overview of the FEA.

4 Discussion

Workshop participants discussed the following topics:

1. The software and systems architectural views have different purposes but also have some overlap. Because an enormous number of views could be built, architecture developers for a system must select the system and software architectural views that are important to them in documenting the architecture. Developers must also specify the order and time sequence for developing those views. Selecting which views to use depends on the purpose for building the architecture, the stakeholders who review the architecture, and other factors, such as those discussed in IEEE Std 1471-2000.

   While everyone agreed that architecture is an essential ingredient in the engineering of non-trivial systems, there was also general agreement that an "architectural storm" is brewing with the many overlapping architectural buzzwords. While it is easy to find references to "information architecture," "enterprise architecture," "system architecture," "system-of-systems architecture," "software architecture," "communications architecture," "hardware architecture," "security architecture," "data architecture," and many other "architectures," it is harder to find crisp definitions of any of them, or descriptions of how they should be used in our engineering discipline. (In fact, one such overlap—"system architecture" versus "software architecture"—can be said to have led to this workshop.)

2. Everyone agreed that, regardless of whether a project deals with a software architecture or a system architecture, views should be built according to the purpose for building them. The group was largely suspicious of any methodology with a closed set of prescribed architectural views. There was some discussion about whether the DoDAF encourages, merely allows, or forbids the use of views other than the three it promotes. The attendees agreed that it would be helpful to clarify the DoD's position on the use of other views.

3. IEEE Std 1471-2000 is a good tool for starting to develop viewpoints. This standard is consistent with the stakeholder/view table in the SEI's Views and Beyond approach.

4. The group identified a number of important uses for the DoDAF views, including

   - as an initial stage in developing a large-scale system. In this case, the evolution from a DoDAF set of views to a set of software architecture views is necessary if the system is software intensive (because software views are needed by the software developers).
   - as a source-selection mechanism for fly-off evaluation. In this case, the appropriate DoDAF views should be chosen, and—if the system is software intensive—some

software architectural views should be built to describe the important software capabilities being proposed by the competing teams.

- as a mechanism for making investment decisions. In this case, since the investment decision is often to "mix and match" among the proposed alternatives, many options are discarded. Once again, if the system is software intensive, some software architectural views should be developed to demonstrate the capabilities that are poorly represented in the DoDAF.

5. The group felt that the DoDAF did not adequately represent architectures that involve software styles such as distributed data or distributed computation; these styles require some software architectural views. They also felt that, while the DoDAF sufficiently addressed broad, overarching designs, it did not adequately capture detailed system design.

6. The end user is under-represented in the development and review of the views or products. For example, the end user has little patience for reviewing hundreds of pages of documents and diagrams. Members of each end-user class, however, can be walked through a number of important use case scenarios that are relevant to the way they will use the system. The end user relates well to demonstrations of capabilities, especially person-in-the-loop prototype simulations. The models for these demonstrations must have reasonable computer-human interfaces.

7. The current DoDAF is representation oriented, and does not impose or recommend a process for architecture development. Such a process can be quite sophisticated and can differ across contractors and vendors. Guidance and expertise can prevent the developer from making mistakes others have already made. Other considerations include the following:

   - There is no obvious way to determine the effect of a reduction in scope, reduction in funding, or advancement in schedule.
   - The "reward" structure is not aligned with the desire to create interoperable constellations of systems. Each system manager is rewarded by the progress of his or her system, and the integration of the constellations of systems becomes secondary.
   - There is no clear set of criteria to determine what constitutes "acceptable and good" versus "unacceptable and poor" for individual view products or the set of products developed.

8. The views in the DoDAF and in the software architecture realm tend to be complex and are often captured using a variety of notational styles.

   - Software architects use the word "view" to describe a set of software elements and the relationships among them. For example, a logical view describes classes and the relationships among them, and a process view describes processes and their uses. The definitions of views are complicated by the fact that more than one representation is possible for each view (e.g., state transition diagrams and statecharts can be used to

represent the behavioral aspects of processes) and that UML-based tool sets support many views and multiple representations for each view.

- The DoDAF discusses framework products that are included in 3 types of views: (1) the Operational View (OV), which contains 7 products; (2) the System View (SV), which contains 11 products; and (3) the Technical View (TV), which contains 2 products.

  Moreover, the DoDAF contains two All-Views (AV) products that do not comprise a separate view but rather include aspects of the architecture that apply to the architecture as a whole (e.g., the AV-2 product is the integrated dictionary for the whole architecture and contains architecture information from all three views).

9. Since both system and software architectures describe elements and how they relate to each other, there is likely to be some conceptual confusion. Therefore, there will probably also be confusion between DoDAF views developed by system engineers and software architecture views developed by software engineers. It is also likely that teams will mistakenly interpret the DoDAF as sufficient to document the software architecture. This is wrong because there is superficial overlap among the following:

   - the logical view of software architecture as defined by RUP [Kruchten 01] and the System Functionality Description product of the DoDAF
   - the process view of software architecture (again, as defined by RUP) and the System State Transition Description product of the DoDAF
   - use cases in the software architecture (defined by RUP as the "plus one" view, and captured by sequence diagrams) and the System Event Trace Description product of the DoDAF
   - the deployment view of software architecture (defined in the Views and Beyond approach summarized in Section 3) and the allocation of the system functions to the systems that implement them in the DoDAF's Systems Interface Description; that product and the supporting Systems-Systems Matrix may also be used to detail the inter-system software interfaces (i.e., what is currently documented in the Interface Description Documents [IDDs]).

10. System architectures (especially as represented by the DoDAF) are particularly concerned with functionality, whereas software architectures are more concerned with achieving functionality that is specified elsewhere. The software is represented by the system functions in the DoDAF; this representation is not appropriate for a software architecture because a software architecture shows how functions are achieved as a result of cooperating structural elements.

    - The system engineer's view of application functionality tends to be oriented toward the domain challenges associated with the function, while the software engineer concentrates on the services provided to achieve the functionality. These approaches can be quite different. For example, a system engineer may be interested in the

development of an aircraft's track based on timed inputs from multiple sensors, whereas the software engineer is likely to be interested in how the tracks get distributed to clients. Both are important, but the mindset for each is different.

- The system engineering community seems to be comfortable with the well-established IDEF approach to detailing the architecture that starts with designing the hardware elements with associated functionality. The software engineering community gave up on the IDEF approach many years ago in favor of an object-oriented approach that allocates software to hardware at a later time in the development cycle. Many of the major software components that are distributed throughout the system are poorly represented by the IDEF approach but are well represented by the object-oriented approach. The software infrastructures, such as operating systems, communications protocols, and distribution middleware, are all poorly represented in the DoDAF approach. The tensions between the two communities make resolving these problems challenging.

11. The interactions of the system with its environment are treated quite differently in the software architecture and the DoDAF. The software architecture relies heavily on use cases to describe how multiple actors (end user or external system) interact with the various automated elements of the system. The DoDAF uses activity diagrams (OV-5) to describe the general interaction between activities conducted at nodes within the system; the DoDAF does not distinguish between manual and automated activities, since this decision is made later. The functions are traced back to the OV-5 diagram relationships captured in the SV-5 diagrams. There is a strong correlation between use cases and the OV-5 and SV-5 diagrams. In addition to the product descriptions and the data element definition tables (which detail the relationships across products), the object-oriented example in the deskbook also provides guidance on these relationships.

12. The tool sets that support the architectures have been inconsistent in the past. For the last 10 years, the software tool development community has been building a UML standard that is targeted at the software architectural views and is the basis for most current graphical tool sets associated with building software architectural views. Additional UML tool support includes the following capabilities, which many software architects use to build their software architecture and design representations:

    - consistency checking between the different views, which is very necessary and which is performed by these tool sets, and is very necessary. It is almost impossible, given hundreds of complicated diagrams and tables, to determine consistency by manual inspection.
    - export and import of representations between tool sets

    Until recently, many of the DoDAF views were not UML compliant, and could not be built, consistency-checked, exported, or imported. The UML-based tools were built initially for software design, rather than software architectures, and hence lack some features that many software architects believe are important.

13. Some parts of the community believe that architecture is shaped more by its quality attributes or "ilities" (performance, availability, modifiability, security, usability, etc.) than by its functionality. Though this is a well-accepted belief in software architecture, there are few such representations in the DoDAF view products.

    - The DoDAF OV3 product asks for information-exchange performance.
    - UML extensions allow for performance annotations.

    However, methods and procedures (such as the Architecture Tradeoff Analysis Method[SM] [ATAM[SM]]) have been developed to analyze software architectures against quality attribute scenarios; these methods can either produce high confidence that the architecture will satisfy its major business drivers, or identify risks, tradeoffs, and concerns. Analysis methods for the DoDAF have not been reported publicly, though they are undoubtedly used by architects in many cases.

14. The chief information officers (CIOs) and the materiel developers mandate the use of standards and commercial products, including middleware such as Java 2 Enterprise Edition (J2EE), .NET, common object request broker architecture (CORBA), and the Web.

    - The DoDAF uses the TV-1 and TV-2 views to represent current and future standards, but relationships between these standards were not shown in the diagrams in the previous version of the DoDAF. To remedy this situation, the DODAF has included relationships between the standards as detailed in the TV-1 and TV-2 views, and the architectural elements to which these standards correspond (e.g., systems as well as software and hardware components of systems).
    - One approach used by software architects is a "layering" view to describe how applications, user interfaces, middleware, computing platforms, sensors, and actuators interact. However, this approach is often depicted weakly in the architecture. Another approach is to have a constraint model that establishes responsibilities and obligations that each component must fulfill to behave predictably.
    - Though the FEA attempts to address the standards and commercial off-the-shelf (COTS) issues explicitly, the representations to capture these issues are not yet defined, making it difficult to judge the representations' effectiveness.
    - The FEA focuses on information technology (IT), which is largely associated with distributed access to large-scale commercial databases and is often concentrated on providing high-volume throughput to serve many customers as quickly as possible. Many DoD systems must handle significant real-time response requirements. In such systems, commercial database management system (DBMS) products are used with

[SM] Architecture Tradeoff Analysis Method and ATAM are service marks of Carnegie Mellon University.

care, in a way that ensures that the COTS product either meets the performance requirement or is only used in the system's non-critical computations.

5 Summary

A summary of the implications of using the approaches described in the previous sections is provided below. All the following statements refer to large-scale, software-intensive systems of systems.

1. The DoDAF and current software architecture approaches have been developed separately, by different organizations, with different purposes, and with little overlap. Hence, there are significant differences between their favored representations, and there is no way to ensure compatibility and consistency among their different views.
2. There is a need to select which views (system and software) will be needed for a system. IEEE Std 1471-2000 and the SEI's Views and Beyond approach (which leads to 1471-compliant documentation) both provide guidance on selecting views.
3. The DoDAF does not represent software architectures; some software architectural views are needed to supplement the DoDAF products to understand how well these systems will operate.
4. None of the views conveniently represents multi-stage transitions from stovepiped legacy systems to interoperable systems of systems, even though this is "where the action is" nowadays in developing mission-critical systems. Some additional approach, such as an MEP, is needed.
5. Currently each system program office (SPO)/contractor combination must struggle—with little guidance—with the differences between the DoDAF and current software architecture approaches to develop individual approaches for solving their problems. They must then train their staffs to follow the approach, using available tool sets as much as possible.

Though there is certainly room for improving this situation, no detailed discussions took place at the workshop. Some of the above issues can be targeted for further workshops.

Appendix A Documenting Software Architectures Using the "Views and Beyond" Approach

Authors' note: The material in this appendix is based on the book Documenting Software Architectures: Views and Beyond *[Clements 02].*

Introduction: Viewtypes, Styles, and Views

Three years ago, researchers at the Software Engineering Institute and the Carnegie Mellon School of Computer Science set out to answer the question: "How should you document an architecture so that others can successfully use it, maintain it, and build a system from it?" The result of that work is an approach we loosely call "views and beyond."

Modern software architecture practice embraces the concept of architectural views. A view is a representation of a set of system elements and the relations associated with them. Views are representations of the many system structures that are present simultaneously in software systems. Modern systems are more than complex enough to make it difficult to grasp them all at once. Instead, we restrict our attention at any one moment to one (or a small number) of the software system's structures that we represent as views.

Some authors prescribe a fixed set of views with which to engineer and communicate an architecture. Rational's Unified Process, for example, is based on Kruchten's 4+1 view approach to software [Kruchten 01]. The Siemens Four Views model [Hofmeister 00] is another example. A recent trend, however, is to recognize that architects should produce whatever views are useful for the system at hand. IEEE Std 1471-2000, a recommended best practice for documenting the architectures of software-intensive systems exemplifies this philosophy [IEEE 00]; it holds that an architecture description consists of a set of views, each of which conforms to a viewpoint, which, in turn, is a realization of the concerns of one or more stakeholders.

This philosophy about views leads to the fundamental principle of the Views and Beyond approach:

> *Documenting an architecture is a matter of documenting the relevant views, and then adding documentation that applies to more than one view.*

What views are available, from which the views relevant to a system can be chosen? Plenty, in fact, too many. To lend some order to an otherwise-chaotic collection of possible views, we find it extremely helpful to think about views in groups, according to the kind of information they carry. Architects carry out their creative task by thinking about the system in three different ways at once:

1. How is the system to be structured as a set of code units?
2. How is the system to be structured as a set of interacting runtime elements?
3. How is the system to relate to non-software structures in its environment?

Considering views along the lines of these three broad categories helps an architect think in naturally structured terms about the system, and helps consumers of documentation discriminate among the separate concerns that an architecture manifests. We call the categories viewtypes. The three viewtypes are:

1. **Module viewtype.** In views belonging to the module viewtype, the elements are modules, which are units of implementation. Modules represent a code-based way of considering the system. Modules are assigned areas of functional responsibility and assigned to teams for implementation. There is less emphasis on how the resulting software manifests itself at runtime. Relations among modules shown in module views include *is a*, *is part of*, and *depends on*.
2. **Component-and-connector viewtype.** In views belonging to the C&C viewtype, the elements are components (which are principal units of computation) and connectors (which are the communication vehicles among components). The principle relation shown in C&C views is attachment between the components and the connectors.
3. **Allocation viewtype.** Views belonging to the allocation viewtype show the relationship between the software elements and elements in one or more external environments (hardware, organizational, environmental, etc.) in which the software is created and executed.

Even within the confines of a viewtype, elements and relation can be specialized in known ways, resulting in styles. Styles represent known design approaches to architectures. In the C&C viewtype, many styles are well known. By restricting the components to interact via a client-server request-reply connector, and by restricting the communication paths among the elements, a client-server style emerges. Or, by restricting the components to be data repositories and data accessors that communicate via connectors that provide the appropriate communication mechanisms, a shared-data style emerges.

Many authors have catalogued C&C styles (e.g., the work of Shaw and Clements [Shaw 97]). However, the other two viewtypes are just as rich with respect to styles. For example, by specializing the relation among modules to "allowed to use" and imposing a strict ordering on the relation, the well-known layers style emerges. Specializing the relation to *is part of*

and modules to elements that have functional responsibilities yields the module decomposition style. Employing the *is a* relation and other constraints yields a generalization style, the basis for inheritance relations in object-oriented systems.

The allocation viewtype can host various styles depending on how the software and environmental elements are specialized. Allocating modules to a development organization's structure produces the work assignment style. Allocating processes to processors defines the deployment style. And allocating modules to a development environment's file structure gives us the implementation style.

When a style is bound to a particular system, the result is a view.

Choosing the Views

Our fundamental principle cited in Section 3 implies that the first task for an architect is to decide which views are relevant. Our approach provides a simple three-step procedure for choosing the views relevant to a particular project's needs. In concert with IEEE Std 1471-2000, it is based on determining the needs of the stakeholders.

Step 1: Produce a Candidate View List

Begin by building a stakeholder/view table for your project. Enumerate the stakeholders for your project's software architecture documentation down the rows. Be as comprehensive as you can. For the columns, enumerate the views that apply to your system. Some views (e.g., decomposition, uses, and work assignment) apply to every system, while others (C&C views, the layered view) only apply to systems designed according to the corresponding styles.

Once you have the rows and columns defined, fill in each cell to describe how much information the stakeholder requires from the view: none, overview only, or detailed information. We encourage architects to hold a workshop with stakeholders or their representatives to begin a dialogue about what information they will need from the documentation.

The candidate view list consists of those views for which some stakeholder has a vested interest.

Step 2: Combine Views

The candidate view list from Step 1 is likely to yield an impractical number of views. Step 2 winnows the list to a manageable size.

First, look for views in the table that require only overview depth, or that serve very few stakeholders. See if the stakeholders could be equally well served by another view having a stronger constituency.

Next, look for views that are good candidates to become combined views. A combined view shows information native to two or more separate views. A rule of thumb is that if there is a strong correspondence between the elements in two views, then they are good candidates to be combined.

Step 3: Prioritize

After Step 2, you should have the minimum set of views needed to serve your stakeholder community. At this point, you need to decide what to do first. For example, some stakeholders' interests supersede others. A project manager or the management of a company with which yours is partnering often demands attention and information early and often, and you may want to cater to his/her needs first.

Documenting a View

The unit of documentation for a view is a view packet, which is the smallest unit of information about the system you would ever want to give a stakeholder. View packets are a mechanism to "chunk" the information in a view into manageable pieces, because a single unit of documentation that portrayed all the information in a view (especially for large and complex systems) would be unmanageably complex. A view packet can show information about a small portion of the system, or it can show information at a particular level of detail. For instance, the first view packet in a view might show the entire system, but with coarse-grained information. Subsequent view packets could show more detail about each element (such as its substructure). View packets let a stakeholder pan and tilt a "camera" of interest around the system in a view; he/she can zoom in or zoom out to/from elements of interest, and jump from view to view in an organized fashion.

No matter the view, the documentation for a view packet is placed into a standard organization or template comprising seven parts:

1. **A primary presentation** shows the elements and relationships among them that populate the portion of the view shown in this view packet. The primary presentation should contain the information you wish to convey about the system (in the vocabulary of that view) first. The primary presentation is usually graphical. If so, it must be accompanied by a key that explains or points to an explanation of the notation.
2. **An element catalog** details those elements (and their properties, including interfaces) depicted in the primary presentation. In addition, if elements or relations relevant to this

view packet were omitted from the primary presentation, the catalog is where they are introduced and explained.

3. **A context diagram** shows how the system (or portion of the system) depicted in the primary presentation relates to its environment.
4. **A variability guide** shows how to exercise any variation points that are part of the architecture shown in this view packet.
5. **An architecture background** or rationale explains why the design reflected in the view packet came to be.
6. **An "other information" section** contains items that vary according to the standard practices of each organization or the needs of the particular project.
7. **Related view packets** provide a pointer to the view packet's parent, siblings, and children (if any). In some cases, a view packet's children may reside in a different view, as when an element in one style (e.g., a filter in a pipe-and-filter view) is decomposed into a set of elements in a different style (e.g., a set of communicating processes).

Documenting Information that Applies to More than One View

The final piece of architecture documentation is the information that applies to more than one view and to the entire package. It ties together the views and provides a holistic picture of the total design. Cross or "beyond-view" documentation consists of the following sections:

1. **Documentation roadmap.** The documentation roadmap is the reader's introduction to the information that the architect has chosen to include in the suite of documentation. A roadmap begins with a brief description of each part of the documentation package. For each view in the package, the roadmap gives a description of the view's element types, relation types, and property types. The roadmap also gives a description of what the view's purpose. The information can be presented by listing the stakeholders who are likely to find the view of interest, and by listing a series of questions that can be answered by examining the view. The roadmap follows with a section describing how various stakeholders might access the package to help address their concerns. This section might include short scenarios such as "a maintainer wishes to know the units of software that are likely to be changed by a proposed modification."
2. **View template.** A view template is the standard organization for a view. Its purpose is to help a reader navigate quickly to a section of interest. It helps a writer organize the information and establish criteria for knowing how much work is left to do.
3. **System overview.** A system overview is a short prose description of what the system's function is, who its users are, and any important background or constraints. The purpose is to provide readers with a consistent mental model of the system and its purpose.

4. **Mapping between views.** Helping a reader or other consumer of the documentation understand the relationship between views will help that reader gain a powerful insight into how the architecture works as a unified conceptual whole.

5. **Directory.** The directory is simply an index of all the elements, relations, and properties that appear in any of the views, along with a pointer to where each one is defined and used.

6. **Project glossary and acronym list.** The glossary and acronym list define terms unique to the system that have special meaning. These lists, if they exist as part of the overall system or project documentation, might be given as pointers in the architecture package.

7. **Cross-view rationale.** This section documents the reasoning behind decisions that apply to more than one view. Prime candidates for cross-view rationale include documentation of background or organizational constraints that led to decisions of system-wide import.

Summary

Adopting a view-based approach to documentation, and then following that approach with discipline, helps the architect design (and then communicate) along clean conceptual lines that are not haphazardly mixed. Readers will be able to digest the information quickly, and see how the system is structured into a set of well-separated but mutually supporting design spaces.

Our approach frees the architect from the confines of a fixed set of views or having to choose from prescriptions that conflict with each other (e.g., the work of Hofmeister and associates [Hofmeister 00] and Kruchten [Kruchten 01]). The architect is free to choose only those views that are appropriate to the system under construction.

To help the architect make that choice, we have also provided a simple three-step procedure for choosing the relevant views for a system based on stakeholders' concerns. This procedure uses the concept of combined views and prioritization to bring the view set into manageable size for real-world projects.

We have also provided a simple but powerful way to categorize views. Structuring views (and hence, architectural documentation) into the three broad categories defined by the module, component-and-connector, and allocation viewtypes provides a strong intellectual handle for producing architectural information, and understanding documentation produced by others. In this light, views can be seen to belong in one of the three viewtypes or be combinations (perhaps unintended) of views in different viewtypes or styles. The result is greater insight.

By recognizing three viewtypes we can expand previous notions of an architectural style to show that module and allocation styles are a consistent conceptual extension to runtime styles and provide a rich framework in which to make architectural decisions.

Appendix B C4ISR Architecture Framework

Authors' Note: The material in this appendix comes from the C4ISR Architecture Framework [C4ISR 97]. That framework is the predecessor of the DoD Architecture Framework, which is currently in progress and therefore not quotable.

Executive Summary

The Framework defines three related views of architecture: operational (OV), systems (SV), and technical standards (TV). Each view is composed of sets of architecture information that are depicted via graphic, tabular, or textual products. The All-DoD Core Architecture Data Model (CADM) defines the data structure and relationship for architecture information.

The Framework is partitioned into two volumes and a deskbook:

- Volume I provides definitions, guidelines, and some background material.
- Volume II contains descriptions of each of the product types.
- The DoD Architecture Framework Deskbook provides supplementary guidance to Framework users.

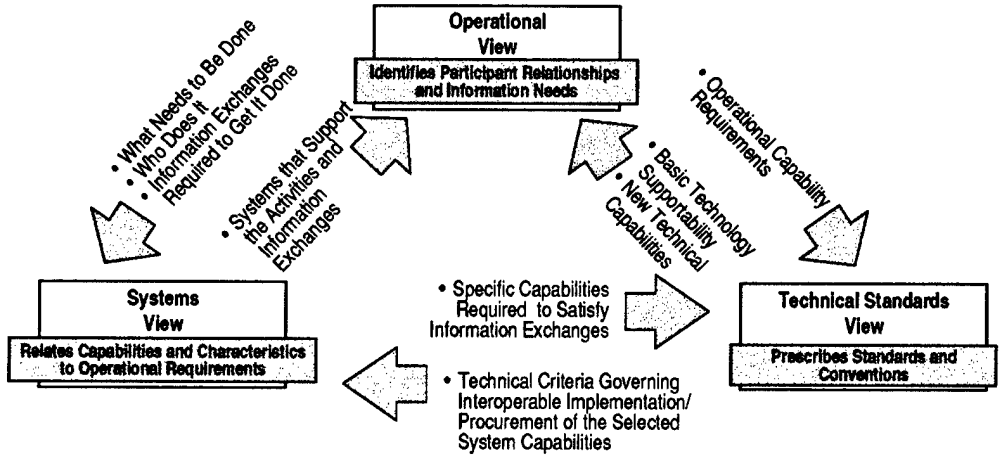


Figure 1: Linkages Among Views

Appendix C IEEE Std 1471-2000

Authors' Note: Text in this appendix comes from IEEE Std 1471-2000, copyright 2000, by IEEE [IEEE 00].

Introduction

(This introduction is not part of IEEE Std 1471-2000, *IEEE Recommended Practice for Architectural Description of Software-Intensive Systems.*)

It has long been recognized that "architecture" has a strong influence over the life cycle of a system. In the past, hardware-related architectural aspects were dominant, whereas software-related architectural integrity, when it existed, was often first to be sacrificed in the course of system development. Today, software-intensive systems are pervasive. The cost of software development and the increasing complexity of software systems have changed the relative balance. Software technology is maturing rapidly. The practice of system development can benefit greatly from adherence to architectural precepts.

However, the concepts of architecture have not been consistently defined and applied within the life cycle of software-intensive systems. Despite significant industrial and research activity in this area, there is no single, accepted framework for codifying architectural thinking, and thereby facilitating the common application and evolution of available and emerging architectural practices.

The IEEE Architecture Planning Group (APG) was formed in August 1995 to address this need. The APG was chartered by the IEEE Software Engineering Standards Committee (SESC) to set a direction for incorporating architectural thinking into IEEE standards. The result of the APG's deliberations was to recommend an IEEE activity with these goals:

- To define useful terms, principles and guidelines for the consistent application of architectural precepts to systems throughout their life cycle;
- To elaborate architectural precepts and their anticipated benefits for software products, systems and aggregated systems ("systems of systems");
- To provide a framework for the collection and consideration of architectural attributes and related information for use in IEEE Standards; and
- To provide a useful road map for the incorporation of architectural precepts in the generation, revision and application of IEEE standards.

In April 1996 SESC created the Architecture Working Group (AWG) to implement those recommendations.

This recommended practice addresses the activities of the creation, analysis, and sustainment of architectures of software-intensive systems, and the recording of such architectures in terms of architectural descriptions. A conceptual framework for architectural description is established. The content of an architectural description is defined. Annexes provide the rationale for key concepts and terminology, the relationships to other standards and examples of usage.

Scope

This standard addresses the architectural description of software-intensive systems. A software-intensive system is any system where software contributes essential influences to the design, construction, deployment and evolution of the system as a whole.

The scope of this standard encompasses those products of system development which capture architectural information. This includes architectural descriptions which are used for:

- the expression of the system and its evolution;
- communication among the system stakeholders;
- evaluation and comparison of architectures in a consistent manner;
- planning, managing, and executing the activities of system development;
- the expression of the persistent characteristics and supporting principles of a system to guide acceptable change;
- the verification of a system implementation's compliance with an architectural description; and,
- recording contributions to the body of knowledge of software-intensive systems architecture.

Purpose

The purpose of this standard is to facilitate the expression and communication of architectures and thereby lay a foundation for quality and cost gains through standardization of elements and practices for architectural description.

Despite significant efforts to improve engineering practices and technologies, software-intensive systems continue to present formidable risks and difficulties in their design, construction, deployment and evolution. Recent attempts to address these difficulties have focused on the earliest period of design decision-making and evaluation, increasingly referred

to as the "architectural level" of system development. The phrases "architectural level" and "architecture" are widely, if imprecisely, used. Their use reflects acceptance of an architectural metaphor in the analysis and development of software-intensive systems. A key premise of this metaphor is that important decisions may be made early in system development in a manner similar to the early decision-making found in the civil architecture profession.

Many innovations are resulting from this attention to the architectural level, among them architectural description languages and associated tools and environments, architectural frameworks, models and patterns, and techniques for architectural analysis, evaluation and architecture-based reuse. While these efforts differ considerably in important aspects, sufficient commonality exists to warrant the development of a recommended practice to codify their common elements.

These innovations are occurring, and maturing, rapidly within many research and application communities, and they reflect differing interests, influences, insights, and intentions. There is a general consensus on the importance of the "architectural level of systems development," and that that level consists of early decision-making about overall design structure, goals, requirements, and development strategies. However, there has not yet emerged any reliable consensus on a precise definition of a system's "architecture," how it should be described, what uses such descriptions may serve, or where and when it should be defined. The boundaries and relationships between architectural trends and practices and other practices, and between architectural technology and other technology, are not yet widely recognized.

In such situations, progress often depends on mediating influences. Potential adopters of architectural practices and technology need a frame of reference within which to address implementation and adoption decisions. Technology developers need a frame of reference within which to communicate the motivating concepts of their technology, and to accumulate and appreciate feedback from early adoption.

To these ends, this standard is intended to reflect generally accepted trends in practices for architectural description and to provide a technical framework for further evolution in this area. Furthermore, it establishes a conceptual framework of concepts and terms of reference within which future developments in system architectural technology can be deployed. This standard codifies those elements on which there is consensus: specifically the use of multiple views, reusable specifications for models within views, and the relation of architecture to system context.

Intended Users

The principal class of users for this standard comprises stakeholders in system development and evolution, including:

- those that use, own and acquire the system (users, operators, and acquirers, or clients),
- those that develop, describe and document architectures (architects),
- those that develop, deliver and maintain the system (architects, designers, programmers, maintainers, testers, domain engineers, quality assurance staff, configuration management staff, suppliers and project managers, or developers), and
- those who oversee and evaluate systems and their development (chief information officers, auditors, independent assessors).

A secondary class of users of this standard comprises those involved in the enterprise-wide, infrastructure activities that span multiple system developments, including: methodologists, process and process improvement engineers, researchers, producers of standards, tool builders and trainers.

Conformance to this Standard

An architectural description conforms to this standard if that description meets the requirements in Clause 5.

Applying IEEE Std 1471-2000 to the DoDAF

The DoD Framework can be interpreted in terms of IEEE Std 1471-2000 and the reference model for architectural descriptions contained in IEEE Std 1471-2000. With relatively small additions to the contents, and some modifications to the approach used to develop DoD Framework products, the resulting DoD Framework description of a architecture can (minimally) conform to IEEE Std 1471-2000.

First it is useful to see how IEEE Std 1471-2000 defines Architecture:

- Architecture: the fundamental organization of a system embodied in its components, their relationships to each other and to the environment and the principles guiding its design and evolution.

where:

- fundamental organization means essential, unifying concepts and principles
- system includes application, system, platform, system-of-systems, enterprise, product line, ...
- environment is developmental, operational, programmatic, ... context of the system

Although the definition of 'architecture' in the DoD Framework does not match this definition, it does not explicitly contradict the standard's definition.

The DoD Framework conforms to the requirements of IEEE Std 1471-2000 on architectural descriptions in the following ways (in that an architectural description that meets the requirements of the DoD Framework would also meet these IEEE Std 1471-2000 requirements):

- Labeling information (author, version, etc)
- Framework products meet IEEE Std 1471-2000 viewpoint definitions
- Conforming architecture definitions must meet framework product definitions

The DoD Framework is missing some significant items required by IEEE Std 1471-2000. To conform to IEEE Std 1471-2000, an architectural description that conforms to the DoD Framework must add the following items to what is already required by the DoD Framework:

- Notion of Stakeholders and Concerns as part of the DoD Framework approach.
- Association of Viewpoints/Views with Stakeholder Concerns "Why am I producing this product?"
- Flexibility in selecting and constructing new Viewpoints.
- Identification/documentation of known inconsistencies.
- Documentation of rationale for the specific architectural choices

Specifically, information can be added to an existing product (e.g. AV-1) or to a new product (AV-"new") to capture stakeholders and concerns for the architecture, the mapping of viewpoints (Framework products) to concerns, and the documentation of known inconsistencies between the various Framework Products included in this description. The requirement of IEEE Std 1471-2000 for capturing rationale can be done either in a separate Framework product (AV-1 or AV-new), or can be included with each Framework product. It would be best to update the Framework to explicitly capture all of the IEEE Std 1471-2000 requirements in the same way for all applications of the DoD Framework.

Note that the DoD Framework does not explicitly prohibit the construction of additional viewpoints/products, but does not explicitly encourage extensions. The DoD Framework should allow for extensions by its users to meet stakeholder requirements not satisfied by the existing Framework products.

Finally, the DoD Framework document could be rewritten using the IEEE Std 1471-2000 definitions and reference model, which would facilitate its integration with other architectural approaches (including software architectural approaches) that apply the IEEE Std 1471-2000 definitions and model.

References

This standard shall be used in conjunction with the following publications. When the following standards are superceded by an approved revision, the revision shall apply.

IEEE Std 100–1996, IEEE Standard Dictionary of Electrical and Electronics Terms.

IEEE Std 610.12–1990, IEEE Standard Glossary of Software Engineering Terminology.

IEEE/EIA Std 12207.0–1996, IEEE/EIA Standard – Industry Implementation of ISO/IEC 12207:1995, Information Technology – Software Life Cycle Processes.

Appendix D Analytic Views Within the DoDAF

It is difficult to build the architecture for a large-scale, software-intensive system, especially if it involves multi-stage deployment to the field in delivery blocks, mixing at each stage: replacement of legacy systems; introduction of new technology; introduction of new capabilities; changes in business processes; experiments with new technologies and operational concepts; and prototype and demonstration development. Of course, changes in budgets and schedules will always force changes in deployment blocks. Neither the software architectures discussed at this workshop nor the DoDAF is especially good at addressing this problem. One suggested approach at the workshop was to use an n-stage spiral development model, with periodically deployable delivery blocks (1 to 2 years) with the following elements:

- a legacy systems inventory, including a catalog of problems associated with the legacy system
- a business drivers document outlining what is to be accomplished to improve the missions' effectiveness and the life-cycle support efficiency, and to manage the identified risks
- an end-stage architecture using a combination of products from the Operational View (OV), Technical View (TV), and System View (SV). This serves as a vision of the architecture of the eventual system.
- a master evolution plan (MEP) that determines what is developed and deployed at each stage, and captures the roadmap changes as they occur
- the need to coordinate the efforts with the independent development of closely related systems, which may be part of the external environment. The worst-case coordination effort is where the system to be coordinated is "external" for the first few delivery stages and then becomes part of the system.
- The materiel developer (system program office [SPO] or program management office [PMO]) should control the OV, which drives the capabilities to be fielded; the materiel developer is heavily influenced by the combat developer. The materiel developer should also control the TV, which drives the new technology introduced and the technology retired; in this case, the materiel developer is heavily influenced by the chief information officer (CIO) and the current set of technical standards. The OV and TV refer to the selected set of products from the DoDAF. The SPO should also adjust the end-stage vision architecture and the MEP to account for programmatic and technical changes.

- The operational community (e.g., operators, warfighters, and other users) owns the OV of the architecture. The acquiring community (e.g., the programming office) owns the development/maintenance of the SV of the architecture. A number of stakeholders (e.g., standards organizations, commercial technology, and interoperability organizations) influence the TV. The OV and the TV constrain, influence, and provide requirements for the SV. As the OV and TV evolve over time, the SV must evolve also. The reality is that the end vision also evolves over time (hopefully more slowly). For this to work, the stakeholder communities must become a team with the perspective that there is one architecture for which each community has significant input. This can become a major problem if the DoD organizes the architecture around a specific architectural view or when organizations build one view independent of the others—the different architectural views are not mutually exclusive.
- The contractors should manage the SV, which influences (along with cost) the development, testing, and deployment plans.
- There must be close cooperation between the materiel developer, the combat developer, the CIO, and the contractor in determining the OV, TV, and SV for the next stage, and upgrading the vision document.
- Each stage should begin with an upgraded vision document and MEP and the detailed architecture from the previous stage, and should build a detailed architecture for the new stage. The key stakeholders must always be identified and be active participants in agreeing what should be done at each stage. This will involve resolving conflicts among the stakeholders.

Appendix E Federal Enterprise Architecture Framework

Authors' Note: The text in this appendix comes from the Federal Enterprise Architecture Framework *[FEA 99].*

Executive Summary

Overview ...

To facilitate efforts to transform the Federal Government into one that is citizen-centered, results-oriented, and market-based, the Office of Management and Budget (OMB) is developing the Federal Enterprise Architecture (FEA), a business-based framework for Government-wide improvement. The FEA is being constructed through a collection of interrelated "reference models" designed to facilitate cross-agency analysis and the identification of duplicative investments, gaps, and opportunities for collaboration within and across Federal Agencies.

This Federal Enterprise Architecture and Business Reference Model is intended for use in analyzing investment in IT and other capital assets. It will also serve as a pilot for the development of a broader architecture that can serve as the foundation for a comprehensive budget and performance reporting system that supports the budget and performance integration initiative.

Summary of Version 1.0

The Business Reference Model (BRM) presented in this document describes the Federal Government's Lines of Business and its services to the citizen – independent of the Agencies, bureaus, and offices that perform these business operations and provide these services. Developed with significant input from civilian Cabinet and other Federal Agencies (work is currently underway to validate those areas of the model relevant to the Department of Defense), the BRM identifies three Business Areas that provide a high-level view of the operations the Federal Government performs – Services to Citizens, Support Delivery of Services, and Internal Operations/Infrastructure. The three Business Areas comprise a total of 35 external and internal Lines of Business – the services and products the Federal Government provides to its citizens; and 137 Sub-Functions – the lower level activities that Federal Agencies perform.

- The **Services to Citizens** Business Area includes the delivery of citizen-focused, public, and collective goods and/or benefits as a service and/or obligation of the Federal Government to the benefit and protection of the nation's general population. This Business Area includes 22 Lines of Business and 82 Sub-Functions.
- The **Support Delivery of Services** Business Area provides the critical policy, programmatic and managerial underpinnings that facilitate the Federal Government's delivery of services to citizens and other Federal, State and local agencies. This Business Area includes 9 Lines of Business and 32 Sub-Functions.
- The **Internal Operations and Infrastructure Business Area** refers to the "back office" support activities that must be performed for the Federal Government to operate effectively. This Business Area includes 4 Lines of Business and 23 Sub-functions.

Other reference models

The BRM serves as the foundation for additional reference models that will be published in the upcoming months – the Performance Reference Model, Data and Information Reference Model, Application-Capability Reference Model and the Technical Reference Model.

- The **Performance Reference Model** will identify a common set of general performance outcomes and metrics that Agencies use to achieve much broader program goals and objectives.
- The **Data and Information Reference Model** will describe, at an aggregate level, the data and information that support program and business line operations. The model will aid in describing the types of interactions and information exchanges that occur between the Federal Government and its various customers, constituencies, and business partners.
- The **Application-Capability Reference Model** will identify and classify horizontal and vertical IT capabilities that support Federal agencies. The model will aid in recommending applications to support the reuse of business components and services across the Federal Government.
- The **Technical Reference Model** provides a hierarchical foundation to describe how technology is supporting the delivery of the application capability. The model will outline the technology elements that collectively support the adoption and implementation of component-based architectures.

Together, the Business and Performance Reference Models will define objectives for Federal Lines of Business, while the other reference models will define how to best allocate resources, technology, and services to meet those objectives.

Managing the Program

The Federal Enterprise Architecture effort will only be successful if a sustainable and repeatable process is established and the roles of all affected stakeholders are clearly defined

and communicated. To manage and coordinate construction of the FEA, and provide a means of participation for all interested parties (e.g., senior Federal agency IT, budget, planning, and procurement officials), OMB established a FEA Program Management Office (PMO). Led by OMB's Chief Technology Officer and the FEA Program Manager, the PMO is driving the development of Component-Based Architectures to support the 24 Presidential Priority E-Government initiatives, the development of the FEA reference models, and the identification of new opportunities for business process and system consolidation to improve the efficiency and effectiveness of the Federal Government. A prime means of communicating its accomplishments to its many stakeholders and customers is the PMO's Website, located at www.feapmo.gov.

The recently chartered Solution Architects Working Group (SAWG) is playing a key role in assisting Federal Agencies with the technical design, development, and deployment of their E-Government initiatives. Through close collaboration with the E-Government initiative teams, the SAWG is providing the leadership and guidance necessary to promote the principles of Component-Based Architectures.

Appendix F Workshop Announcement

Where: SEI Ballston Office, near Washington, DC

When: January 30, 2003, 1 p.m. - 5 p.m.

Background

The DoD Architecture Framework (DoDAF) is being mandated by the DoD as the basis for building representations of large-scale "systems of systems" (note that it was previously called the C4ISR Architecture Framework). These representations cover the operational, systems, and technical architectures—each as a set of interrelated views.

The SEI has extensive experience with software architecture, and has developed approaches to documenting, building, and analyzing software architectures. SEI staff members have published many papers and books on the subject and performed analysis on many software architectures using the Architecture Tradeoff Analysis Method (ATAM).

During the development life cycle, DoDAF views serve as a basis for the development of a software architecture, but there is no accepted way of doing this. This workshop is a first step to understand the challenges involved in the transformation from DoDAF representations to software architectural representations.

The agenda for the workshop is shown below.

1:00 to 1:15	Bill Wood (SEI)	Introductions
1:15 to 1:45	Paul Clements (SEI)	Software Architectural Representations
1:45 to 2:00	Dave Emery (MITRE)	
2:00 to 2:15	Loring Bernhardt (MITRE)	Practical Ways of Viewing Software Architecture Within the DoD Architecture Framework
2:15 to 2:45	Fatma Dandashi (MITRE)	DoD Architecture Framework V1.0 Update
2:45 to 3:00	Break	
3:00 to 3:30	John Weiler (IAC EA)	OMB A130 Guidance
3:30 to 4:45	Bill Wood	Facilitated Discussion

Appendix G Biographies of Authors

William G. Wood, SEI

William Wood has been a member of the technical staff at the SEI at Carnegie Mellon University for 18 years. During this time, he has managed a technical program and technical projects, and provided technical support to the program development organization. He is currently working in software architecture with a number of clients. Previously Wood had worked in process control automation for Westinghouse Electric Corp. for 20 years. He has an MSEE from Carnegie Mellon University and a B.Sc. in Physics from Glasgow University, Scotland.

Paul Clements, SEI

Dr. Paul Clements is a senior member of the technical staff at Carnegie Mellon University's SEI, where he has worked for nine years leading or coleading projects in software architecture documentation and analysis, and software product line engineering. Clements is coauthor of three practitioner-oriented books about software architecture: *Software Architecture in Practice*, *Evaluating Software Architectures: Methods and Case Studies*, and *Documenting Software Architectures: View and Beyond*. He also coauthored *Software Product Lines: Practices and Patterns* in 2001. He was coauthor and editor of *Constructing Superior Software*, and has written dozens of papers in software engineering that reflect his interest in the design and specification of challenging software systems.

John Weiler, Interoperability Clearinghouse

John Weiler is currently the Executive Director and cofounder of the Interoperability Clearinghouse (ICHnet.org), a public-private partnership formed to capture, normalize, and share e-business best practices for architectures and interoperability. He has over 25 years of experience in systems engineering, configuration management, and applied architectures.

His current efforts are focused on emerging enterprise architecture methods and tools in some of the most progressive institutions, and he has been a leading force in advancing architectures and interoperability practices in government and industry.

Weiler is a frequent distinguished speaker at conferences, workshops, and executive training programs in the U.S. and abroad. Weiler is a 1978 graduate of the University of Maryland,

School of Business (Senatorial Scholarship) specializing in Information Systems Management and Statistics.

Huei-Wan Ang, The MITRE Corporation

Huei-Wan Ang is a senior software systems engineer with the MITRE Corporation. She has been working on the DoD Architecture Framework since October 2001. She has 10 years of experience in object-oriented software development and software architecture. She has an MS in Information Systems from the American University and a BS in Computer Science from George Mason University.

Fatma Dandashi, The MITRE Corporation

Dr. Fatma Dandashi has been working on the DoD Architecture Framework as a member of the MITRE development team supporting the Office of the Secretary of Defense (OSD) since she joined MITRE in 1999. Her major contributions to the draft revision titled *DoD Architecture Framework V2.1* and the later draft revision entitled *DoD Architecture Framework V1.0* consisted of providing a description and representation of the framework products (and their associated architecture information) in UML notation. Since October 2002, Dr. Dandashi has been the task lead for the MITRE development team responsible for revising and publishing the current draft DoD Architecture Framework V1.0, dated January 15, 2003 (Volumes I and II). This draft has been reviewed by the Architecture Framework Working Group and is currently being reviewed by the larger DoD community. An official release of a final DoD Architecture Framework V 1.0 by the OSD is scheduled for July 2003.

Dr. Dandashi holds a PhD in Information Technology from George Mason University, an MS in Computer Science from the University of Louisiana (Lafayette), and a BA in Computers/Business Administration from the Lebanese American University.

Dave Emery, The MITRE Corporation

David Emery is a principal engineer with the MITRE Corporation. He has been working on software architecture and system architecture concepts since 1992. He served on the committee that wrote IEEE Std 1471-2000, *Recommended Practice for Architectural Description of Software-Intensive Systems.*

Mario Barbacci, SEI

Mario Barbacci is a senior member of the staff at the SEI. He was one of the founders of the SEI, where he has served in several technical and managerial positions, including project leader (Distributed Systems), program director (Real-Time Distributed Systems, Product Attribute Engineering), and associate director (Technology Exploration Department). Prior to joining the SEI, he was a member of the faculty in the School of Computer Science at

Carnegie Mellon University. His current research interests are in the areas of software architecture and distributed systems. He has written numerous books, articles, and technical reports, and has contributed to books and encyclopedias on subjects of technical interest.

Steve Palmquist, SEI

Steven Palmquist is a registered professional engineer and a certified project management professional. Before joining the SEI in 2000, he served for 20 years as a program manager and helicopter pilot in the U.S. Coast Guard. His final tour was as the assistant program manager for C4ISR on the Integrated DeepWater System, designated a government reinvestion lab, where he was one of the principal architects of the program's structure. A commercial-rated pilot and a naval aviator, he holds an MSEE from the Naval Postgraduate School and is a graduate of the National Test Pilot School (Avionics).

Sarah Sheard, Software Productivity Consortium

Sarah Sheard has worked in systems engineering and process improvement for over 20 years and is currently the chief technologist leading the systems engineering effort at the Software Productivity Consortium. She received the 2002 INCOSE Founder's Award for her work in INCOSE, including publishing over 20 symposium papers. She led a two-day systems architecture workshop at the Consortium in March and will be a lead author on Consortium architecture products in 2003 and 2004.

Lyn Uzzle, Software Productivity Consortium

Lyn Uzzle is a senior member of the technical staff at the Software Productivity Consortium. She has over 20 years of software/system development and process improvement experience with the Consortium and defense, aerospace, and consulting organizations. Uzzle has a BS in Computer Science from North Carolina State University.

References

[Bass 98] Bass, L.; Clements, P.; & Kazman, R. *Software Architecture in Practice.* Boston, MA: Addison Wesley, 1998.

[Bosch 00] Bosch, J. *Design and Use of Software Architectures.* London: Addison Wesley, 2000.

[Buschmann 96] Buschmann, F.; Meunier, R.; Rohnert, H.; Sommerlad, P.; & Stal, M. *Pattern-Oriented Software Architecture, Volume 1: A System of Patterns.* New York: Wiley, 1996.

[Clements 01] Clements, P.; Kazman, R.; & Klein, M. *Evaluating Software Architectures: Methods and Case Studies.* Boston, MA: Addison-Wesley, 2001.

[Clements 02] Clements, P.; Bachmann, F.; Bass, L.; Garlan, D.; Ivers, J.; Little, R.; Nord, R.; & Stafford, J. *Documenting Software Architectures: Views and Beyond.* Boston, MA: Addison-Wesley, 2002.

[C4ISR 97] Office of the Secretary of Defense Working Group. *C4ISR Architecture Framework, Version 2.0.* Washington, DC, 1997.

[FEA 99] Federal Chief Information Officers' Council. *Federal Enterprise Architecture Framework, Version 1.1.* Washington, DC, 1999.

[Hofmeister 00] Hofmeister, C.; Nord, R.; & Soni, D. *Applied Software Architecture.* Boston, MA: Addison-Wesley, 2000.

[IEEE 00] Institute of Electrical and Electronics Engineers. *IEEE Std 1471-2000.* Piscataway, NJ: IEEE Computer Press, 2000.

[Kruchten 01] Kruchten, P. *The Rational Unified Process: An Introduction*, 2nd ed. Boston, MA: Addison-Wesley, 2001.

[Putman 00] Putman, J. *Architecting with RM-ODP.* Upper Saddle River, NJ: Prentice-Hall, 2000.

[Shaw 97] Shaw, M. & Clements, P. "A Field Guide to Boxology: Preliminary

Classification of Architectural Styles for Software Systems," 6-13. *Proceedings of First International Computer Software and Applications Conference (COMPSAC97)*. Washington, DC, August 11-15, 1997. Piscataway, NJ: IEEE Computer Society Press, 1997. <http://www2.cs.cmu.edu/afs/cs.cmu.edu/project/vit/www/paper_abstracts/Boxology.html>

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
(Leave Blank)	March 2003	Final

4. TITLE AND SUBTITLE	5. FUNDING NUMBERS
DoD Architecture Framework and Software Architecture Workshop Report	F19628-00-C-0003

6. AUTHOR(S)

William G. Wood, Mario Barbacci, Paul Clements, Steve Palmquist, Huei-Wan Ang, Loring Bernhardt, Fatma Dandashi, David Emery, Sarah Sheard, Lyn Uzzle, John Weiler, Art Krummenoehl

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
Software Engineering Institute Carnegie Mellon University Pittsburgh, PA 15213	CMU/SEI-2003-TN-006

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSORING/MONITORING AGENCY REPORT NUMBER
HQ ESC/XPK 5 Eglin Street Hanscom AFB, MA 01731-2116	

11. SUPPLEMENTARY NOTES

12A DISTRIBUTION/AVAILABILITY STATEMENT	12B DISTRIBUTION CODE
Unclassified/Unlimited, DTIC, NTIS	

13. ABSTRACT (MAXIMUM 200 WORDS)

During the Software Engineering Institute's Workshop on the Department of Defense Architecture Framework and Software Architecture, participants from government, industry, and academia discussed the similarities and differences between system and software architecture representations, and how these representations relate with one another. This technical note summarizes the activities of that workshop.

14. SUBJECT TERMS	15. NUMBER OF PAGES
software architecture, system architecture, view, view products	45

16. PRICE CODE

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