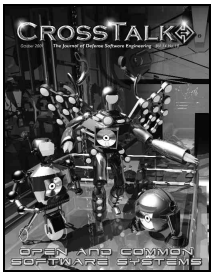

CROSSTALK
October 2001
The Journal of Defense Software Engineering
Vol. 14 No. 10
OPEN AND COMMON SOFTWARE SYSTEMS

Open and Common Software Systems



On The Cover

Kent Bingham, Digital Illustration and Design, is a self-taught graphic artist/designer who freelances print and Web design projects. Robot design concepts by Drew Bingham.

Software Engineering Technology

Departments

CrossTalk

Sponsor	Lt. Col. Glenn A. Palmer
Publisher	Tracy Stauder
Associate Publisher	Elizabeth Starrett
Managing Editor	Pam Bowers
Associate Editor	Benjamin Facer
Article Coordinator	Nicole Kentta
Creative Services Coordinator	Janna Kay Jensen
Phone	(801) 586-0095
Fax	(801) 777-5633
E-mail	crosstalk.staff@hill.af.mil
CrossTalk Online	www.stsc.hill.af.mil/crosstalk/crostalk.html
CRSIP Online	www.crsip.hill.af.mil

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail or use the form on p. 9.

Ogden ALC/TISE
7278 Fourth St.
Hill AFB, UT 84056-5205

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CrossTalk editorial board prior to publication. Please follow the Author Guidelines, available at www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf. CrossTalk does not pay for submissions. Articles published in CrossTalk remain the property of the authors and may be submitted to other publications.

Reprints and Permissions: Requests for reprints must be requested from the author or the copyright holder. Please coordinate your request with CrossTalk.

Trademarks and Endorsements: This DoD journal is an authorized publication for members of the Department of Defense. Contents of CrossTalk are not necessarily the official views of, or endorsed by, the government, the Department of Defense, or the Software Technology Support Center. All product names referenced in this issue are trademarks of their companies.

Coming Events: We often list conferences, seminars, symposiums, etc. that are of interest to our readers. There is no fee for this service, but we must receive the information at least 90 days before registration. Send an announcement to the CrossTalk Editorial Department.

STSC Online Services: www.stsc.hill.af.mil

Call: (801) 777-7026, E-mail: randy.schreifels@hill.af.mil

Back Issues Available: The STSC sometimes has extra copies of back issues of CrossTalk available free of charge.

The Software Technology Support Center was established at Ogden Air Logistics Center (AFMC) by Headquarters U.S. Air Force to help Air Force software organizations identify, evaluate, and adopt technologies to improve the quality of their software products, efficiency in producing them, and their ability to accurately predict the cost and schedule of their delivery.

How Wide Open Should 'Open Systems' Be?

In the early 90s, the concept of open systems became prominent. In 1995, the Software Engineering Institute conducted the conference "Open Systems: Promises and the Pitfalls," where basic definitions, elements of an open-systems approach, and reference models and architectures were presented. After six years, study and debate continues in Department of Defense (DoD) circles on implementation of open systems and the road ahead.

From a philosophical standpoint, one can see the origins of a natural tension between those promoting open systems and those who demand interoperability. The open-systems mindset group dreams of using commercial-based components and standard interfaces common across platforms with *plug-and-play* ability. This model is certainly consistent with DoD's drive for a single industrial base and acquisition reform initiatives aimed at affordability. However, interoperability has historically been achieved by imposing standards for products. Programs must weld these sometimes competing initiatives to both achieve affordability through open systems, and to comply with interface standards and conventions necessary for interoperability. This issue provides a topical primer for the uninitiated and some updates and references for those already in the *briar patch* of defining software architectures for real-time defense systems.

In *Joint Technical Architecture: Impact on DoD Programs*, Judy Kerner of The Aerospace Corporation discusses the motivation for the Joint Technical Architecture (JTA) and the role of interface standards and open-systems architectures in achieving interoperability. She contrasts the JTA to the Defense Information Infrastructure (DII) Common Operating Environment (COE), a related initiative with which it is often confused.

Doug Gardner of Defense Information Systems Agency in *The DII COE: Basic Principles and Future Challenges* describes the advantages and challenges of using the DII COE along the lines of its four basic principles: interoperability, security, customer focus, and best value. He also describes challenges for DII COE in the future and describes the widening *expectations gap* by users who see new capabilities in the commercial marketplace that are still years away from being systematically deployed in the DoD.

In *The DII COE: An Enterprise Framework,* Dr. Gregory Frazier describes the history and architecture of DII COE. He observes that the commonality of the COE rests on the fact that mission applications use it to provide common functionality. When segments provide their own implementation of functions already in the COE, no savings due to reduced maintenance, reuse, or technology insertion are achieved. He also describes how systems' compliance with the COE is measured, and outlines the challenges for programs adopting COE.

In *DII COE for Real-Time: Becoming Reality*, members of the DII COE Real-Time Team provide a status on their work to develop a set of extensions to the existing DII COE capabilities. Lt. Col. Lucie M.J. Robillard, U. S. Air Force; Dr. H. Rebecca Callison and Marilynn B. Goo, The Boeing Company; and John Maurer, The MITRE Corporation provide updates on the several real-time products available for use in 2001 that are part of DII COE. These include a configurable real-time kernel that is hosted on operating systems that have scheduling capabilities and services required for real-time applications.

I've heard that most people read magazines from back to front. That may be a good choice in this month's issue. Ingmar Ögren, partner and chairman of the board for Tofs Inc. and Romet, in Sweden, reminds us of the importance of requirements development starting from a systems mission and capabilities. His article, *Mission-Based Incremental Development of C2 Systems for More Efficient Business Support*, describes how to use modeling for incremental development of C2 systems and maintain consistency between system simulations and product design.

We hope this issue of CROSSTALK benefits you by capturing the recent history and current state of open and common systems within DoD. We know the future direction is still being charted. The prestigious National Research Council is slated to deliver their findings and recommendations later this year, and the Office of the Secretary of Defense Joint Task Force on Open Systems has several pilot programs and demonstration efforts underway. Look for an article from DoD leadership on future directions for open systems later this year.

Glenn A. Palmer

Lt Col Glenn A. Palmer
Director, Computer Resources Support Improvement Program

Joint Technical Architecture: Impact on Department of Defense Programs

Judy Kerner
The Aerospace Corporation

The Department of Defense (DoD) Joint Technical Architecture (JTA) is intended to help achieve weapon systems interoperability and an open systems approach to weapons-system design. This article provides information a DoD program manager, development contractor, system architect, or other JTA stakeholder will need to know to begin applying JTA in system development. This article describes the organization and content of the JTA very briefly, and contrasts it with the Defense Information Infrastructure (DII) Common Operating Environment (COE), a related initiative with which it is often confused. Finally, it describes some of the actions DoD programs must take in order to comply with the mandate for JTA and identifies a few of the additional actions necessary to achieve system interoperability.

In today's increasingly dynamic battlespace, systems that were never intended to work together are often involved in aspects of the same mission, sometimes even deployed in the same tent. In this environment, interoperability (i.e., the ability of systems to exchange information and use common information) is at a premium, but it rarely happens by accident. The Department of Defense (DoD) has begun a number of initiatives to address aspects of this problem.

In the information technology (IT) arena, the DoD Joint Technical Architecture (JTA) [1] is intended to help achieve weapon systems interoperability and to support affordability and an open systems approach to weapon-system design. To accomplish this, the JTA specifies a set of primarily commercial specifications, standards, and guidelines in the areas of information processing, information transfer, modeling, message format, user interface, and security. DoD requires these standards and guidelines to be applied to all new and all changes to DoD information technology and national security systems.

There is a great deal to be said about the JTA, its development and context, related initiatives, and the role of interface standards and open system architectures in achieving interoperability; far too much to cover in one article. The scope of this article is limited to information a DoD program manager, development contractor, system architect, or other JTA stakeholder will need to know to become sufficiently familiar with the JTA to begin applying it in system development. This article discusses the motivation for JTA and quotes from some of the current DoD policy that mandates its use. It describes the organization and content of the JTA very briefly, and contrasts it with the Defense Information Infrastructure (DII) Common Operating Environment (COE), a related initiative with which it is often confused. Finally, it describes some of the actions DoD program personnel must take in order to comply with the mandate for JTA, and identifies a few of the additional actions necessary to achieve system interoperability.

> "It is no longer possible to identify in advance all the systems with which a new system will need to interoperate even in the near term."

Motivation for JTA

The battlefield environment has changed; today, task forces are formed and dissolved in real time to meet dynamic requirements. It is no longer possible to identify in advance all the systems with which a new system will need to interoperate even in the near term. The interfaces between two or more systems have traditionally been defined in Interface Control Documents agreed to by all involved parties. But when the specific combinations of interoperating systems are not known a priori, this approach can become unworkable. The rapid pace of change in the commercial world complicates the situation still further, since increasingly many of the components of DoD systems are of commercial origin. This dynamic environment favors systems that can evolve most easily to meet changing requirements and environments, systems whose interfaces facilitate this rapid flexibility and adaptability.

Both in industry and in DoD, interface standardization and open systems are being used to facilitate this flexibility. The concept is that if a system is implemented with a standard interface, then it should be able to interface at least with other (perhaps unspecified) systems built to use the same standard interface. This approach is well understood for hardware interfaces, as for example, with electrical sockets and plugs. The DoD is moving toward interface standardization and open systems to help achieve the necessary battlefield interoperability.

According to the DoD Open Systems Joint Task Force (OS-JTF) [2], an open system is a "system that implements sufficient open standards for interfaces, services, and supporting formats to enable properly engineered components to be utilized across a wide range of systems with minimal changes, to interoperate with other components on local and remote systems, and to interact with users in a style that facilitates portability." A key characteristic of an open system is that it has standard interfaces that facilitate portability and interoperability of system components, as well as user portability. The JTA and the DII COE are two of the initiatives aimed at increasing this standardization and commonality within the DoD.

JTA Scope and Evolution

Since August 1996 when JTA Version 1.0 [3] was released, JTA's scope of applicability has broadened considerably. Corresponding to the release of JTA Version 1.0,

the Office of the Secretary of Defense (OSD) mandated the JTA for all command, control, communications, computers, and intelligence (C4I) systems and the interfaces of other key assets with C4I systems [4]. JTA Version 2.0 [5] was released in May 1998, and with its implementation memo in November 1998 [6], the scope of application broadened.

The memo said, in part: "JTA, that is the use of applicable JTA standards, is required for all emerging or changes to an existing capability that produces, uses, or exchanges information in any form electronically; crosses a functional or DoD Component[1] boundary; and gives the warfighter or DoD decision maker an operational capability." Waivers from compliance with JTA standards were possible for cost, schedule, or performance impacts, but required approval of the DoD Component Acquisition Executive (CAE) or cognizant OSD authority. Each individual DoD Component was made responsible for implementing the JTA mandate, including compliance assurance, programming and budgeting of resources, and scheduling.

JTA Version 3.0 [7] was released in November 1999; the memo implementing it [8] included the JTA Version 2.0 implementation memo as an attachment, and indicated that the key paragraphs, including those described above, continue to apply. A concern arose that the long time between releases of the JTA might not allow it to keep pace with rapidly changing technology and program needs. So it was decided to allow interim versions of the JTA to be released without new implementation memos, under the condition that the only differences involve movement of standards within the document, from "emerging" status to "mandated" status. A change of this sort precipitated the release of JTA Version 3.1 in March 2000 [9]; the only significant difference between the versions was that in Version 3.1, Gigabit Ethernet was listed as a mandated standard, while in Version 3.0 it had been classified as an emerging standard.

DoD has begun incorporating JTA compliance in major policy documents, which have further broadened its scope of applicability. For example, in May 2000, the chairman of the Joint Chiefs of Staff (CJCS) issued CJCS Instruction 6212.01B [10], which stated the following: "National Security Systems and Information Technology Systems must comply with applicable IT standards contained in the current DoD JTA Service and Agency-specific implementation." DoD Regulation 5000.2-R [11], dated June 2001, stated that "JTA is required for all new or changes to existing IT, including [National Security Systems] NSS," and that "if the use of a JTA mandated standard will negatively impact cost, schedule, or performance, a DoD CAE or cognizant OSD [Principal Staff Assistant] PSA may grant a waiver from use." For mission critical or mission essential programs, all granted waivers must be submitted for review to still higher levels in OSD. All waiver requests are required to detail the cost, schedule, and performance impacts if the waiver is not granted.

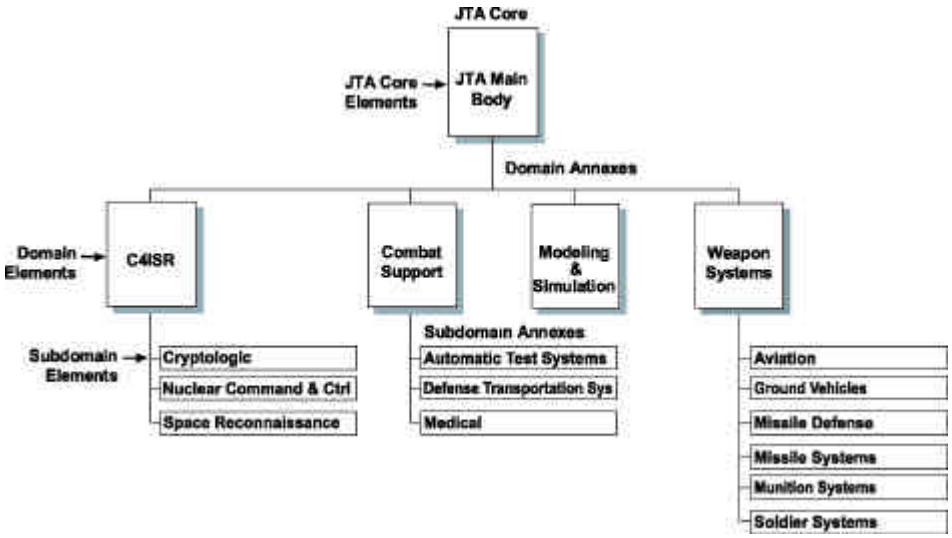


Figure 1: *Joint Technical Architecture Version 4.0 Hierarchy Model*

Policy statements such as these clearly indicate DoD's intent for JTA to be implemented; waivers are allowed if justified, but have to be approved at a very high level. JTA continues to evolve: A "final" JTA Version 4.0 became available April 2001, and the multi-phase review process for JTA Version 5.0 is already in progress.

What is JTA?

The Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR) Architecture Framework Version 2.0 [12] defines three kinds of architecture views for DoD systems. The three views defined are operational architecture (OA), systems architecture (SA), and technical architecture (TA) views. A technical architecture is defined as "the minimal set of rules governing the arrangement, interaction, and interdependence of system parts or elements, whose purpose is to ensure that a conformant system satisfies a specified set of requirements." The DoD JTA is such a technical architecture; it achieves its purpose by identifying the interface standards and conventions necessary for DoD to facilitate information technology interoperability. These standards and conventions facilitate interoperable implementation of the system capabilities described in the SA view, within the operational context described in the OA view.

The structure of the JTA document includes a *core*, four *domain* annexes, and a number of *subdomain* annexes. Figure 1, taken from DoD JTA Version 4.0 [1], shows the hierarchical structure of the JTA and identifies the JTA core, domains, and subdomains.

The JTA core contains common interfaces and standards considered to be applicable to all DoD systems to support interoperability. Domains are intended to identify families of systems. To further support interoperability among the systems of each JTA domain, the corresponding JTA domain annex contains domain-specific JTA standards that are applicable (in addition to those in the JTA core) to the systems of the domain. Similarly, subdomains identify smaller groupings of similar or related systems within a domain; systems within a subdomain must comply with all relevant standards in the JTA core, in the annex for the parent domain, and in the relevant subdomain.

JTA Version 4.0 Structure

The JTA core is divided into sections that contain different kinds of IT standards and guidelines. All of the specifications that are cited as "mandated" in the JTA must enhance interoperability, be technically mature, implementable, and publicly available. The JTA also lists additional standards as "emerging;" their criteria for inclusion are less strict, and they are considered either for elevation to mandated status or for deletion each time the JTA is revised.

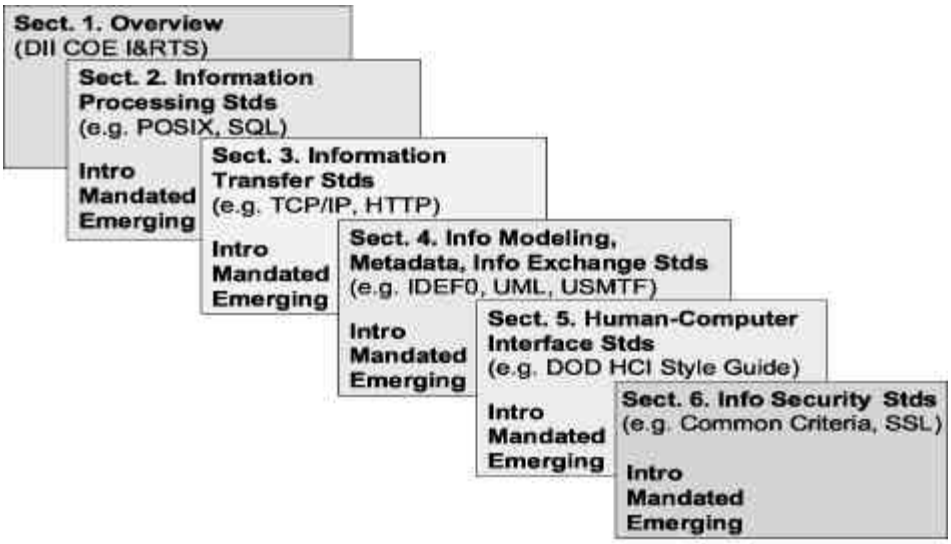


Figure 2: *Structure of the Joint Technical Architecture Core*

Figure 2 shows graphically the structure of the JTA core, with examples of the kinds of standards in each section.

JTA Version 4.0 Section 1 contains an overview of the document and describes a number of related initiatives, including the C4ISR Architecture Framework [12] referred to earlier and the DoD Technical Reference Model (TRM) [13]. It also contains the only policy statements in the JTA itself. In JTA Version 4.0, a new subsection called Policy was introduced into Section 1. One subsection under Policy identifies four key documents[2] applicable for Combined and Coalition Standardization and/or Interoperability, and another subsection mandates use of the DII COE. The remainder of the JTA specifies mandated and emerging information technology standards that are to be complied with whenever applicable.

Compliance with the DII COE is mandated in JTA Section 1 for Command and Control (C2), Combat Support (CS), and Intelligence Systems supporting the Joint Task Forces (JTFs) and Combatant Commands. DII COE is implemented by a set of modular software that provides generic functions or services that are accessed by other software through standard application program interfaces (APIs). DII COE and the levels of DII COE compliance are defined in the DII COE Integration and Runtime Specification (I&RTS) [14], which is identified as one of the mandated standards in the JTA. The JTA further requires that all applications of a system that must be integrated into a DII platform be at least DII COE I&RTS Level 5 compliant with a goal of achieving Level 8. The levels of DII COE compliance are beyond the scope of this article, but as a quick reference, for Level 5 compliance, the system's software would need to be segmented, use the DII COE Kernel, and be installed via COE tools. A brief comparison of JTA and DII COE is presented later in this article. Additional information about DII COE is available in the I&RTS and on the DII COE Web site[3].

JTA core Sections 2 through 6, and the domain and subdomain annexes, contain mandated and emerging information technology standards with brief descriptions and some guidance on when each would apply. Following is an abbreviated discussion of the kinds of standards in each core section, and a very few examples of the standards in the domain annexes. The JTA is available on the Web[4], and the reader is encouraged to browse through the JTA for more information and to look for standards of interest. The standards in these sections of JTA are organized loosely according to the service areas and services defined in the DoD TRM [13].

JTA Section 2 contains standards in a category called Information Processing. These are common software and information technology interface standards such as Portable Operating System Interface (POSIX), Motif, Structured Query Language (SQL), and Common Object Request Broker Architecture (CORBA); some data interchange standards, such as Graphics Interchange Format (GIF), Joint Photographic Experts Group (JPEG), and National Imagery Transmission Format (NITF); as well as some of the more widely used *markup language* standards, such as Hypertext Markup Language (HTML) and eXtensible Markup Language (XML). Many of the standards in this section are so prevalent it is hard to find a commercial product to which one of these standards applies that does not comply with that standard.

JTA Section 3 standards are categorized as Information Transfer Standards. These standards include Internet protocols, e-mail, and networking standards. The standards in this section include Simple Mail Transfer Protocol (SMTP), Multipurpose Internet Mail Extension (MIME), File Transfer Protocol (FTP), Hypertext Transfer Protocol (HTTP), Uniform Resource Locator (URL), and Transmission Control Protocol (TCP)/Internet Protocol (IP). Again, many of these standards are virtually ubiquitous, especially among commercial products. Also in this section are a small number of military standards for which there is no commercial alternative, such as Global Positioning System (GPS) and Military Satellite Communications (MILSATCOM) standards. But as for all of the mandated standards in JTA, these are included only if they are publicly available and widely implemented.

JTA Section 4 is titled Information Modeling, Metadata, and Information Exchange Standards. It includes standards in all three categories. There are modeling

	JTA	DII COE
Contents	Industry and some military specifications and standards	Middleware and infrastructure software and utilities
Features	Interface specifications	Mostly open system products
Software	No software is identified except DII COE	Implemented using DISA-approved COTS and GOTS software
Implementation Context	Compliance with any standard required only if corresponding service is in system; JTA has additional applicability guidance for each standard	I&RTS defines DII COE compliance levels and segmentation, provides rules for interaction among software components
Mandate	Mandated in DoD and DoD Component policies	Mandated in JTA, only for C2, combat support, and intelligence systems

Table 1: *Joint Technical Architecture and Defense Information Infrastructure Common Operating Environment Compared*

standards like Integration Definition (IDEF0), IDEF1X, and Unified Modeling Language (UML); data definition standards such as Defense Data Dictionary System (DDDS); and message formats for information exchange, like Tactical Digital Information Link (TADIL-J) and United States Message Text Format (USMTF).

JTA Section 5 contains Human-Computer Interface (HCI) Standards, including DoD, Motif, and Windows style guides; human-centered design processes; and military symbology standards.

The final section in the JTA core is Section 6, which contains Information Security Standards for various means of protecting confidentiality and integrity of information. Examples include the FORTEZZA Cryptologic Standard; Secure Sockets Layer (SSL) protocol; secure versions of standards that appear in other sections, such as Secure MIME (S/MIME) for encrypted e-mail; and the Common Criteria for evaluation of the strength and functional correctness of Information Assurance products.

The domain and subdomain annexes contain standards that are considered to apply only to specific families of systems so that, for example, the C4ISR Domain includes NITF Extensions, the Modeling & Simulation Domain includes High-Level Architecture (HLA), and the Combat Support domain includes Continuous Acquisition and Life Cycle Support (CALS).

High-Level Comparison of JTA and DII COE

Confusion about the relationship between JTA and DII COE often provokes questions: Is JTA a superset or a subset of DII COE? Can the mandated compliance with JTA be achieved by implementing a system using DII COE? Is selecting a platform that does not support DII COE sufficient grounds for a JTA waiver? To respond simply, the answer to all these questions is "No." JTA mandates the use of DII COE for certain systems, but complying with JTA means complying with all applicable JTA standards; DII COE implementation does not imply JTA compliance (although it may help, since most DII COE products are also JTA-compliant). Table 1 above contrasts JTA and DII COE.

Program personnel must understand the difference between requirements for JTA compliance and for DII COE compliance. Here are some important points to remember:

- JTA and DII COE compliance are not the same. If a program is required to comply with JTA, then implementing DII COE may also be necessary (i.e., for command and control, combat support, and intelligence systems). However, the relevant JTA standards must still be identified, and the system assessed for compliance with them.
- The scope and application are broader for JTA. DoD policy mandates JTA for all national security systems and IT systems. JTA mandates DII COE compliance only for command and control, combat support, and intelligence systems.
- The impact on program architecture may be greater for DII COE, because it contains software that must be incorporated into the system architecture. But JTA standards may also drive some aspects of the system architecture – it is important to develop a JTA profile while the architectural impact can be minimized.

Complying with JTA

OSD mandates that compliance with all applicable JTA standards must be considered for all new programs and changes to existing programs. What does this mean for a program? JTA contains many industry standards that will be implemented regardless of the mandate, so for those parts of a system, there will be no impact at all. For many other parts of the system, if the JTA standards are kept in mind during the initial design of the system, then when there are architectural decisions to be made that could make JTA compliance either trivial or difficult to accomplish, the decision can be made to go towards JTA compliance without additional cost. As was mentioned earlier, it is important to remember that DII COE compliance at any level is not sufficient to ensure JTA compliance, even though DII COE compliance is also required for many programs.

The applicable mandated standards in the JTA are expected to be used as the starting set of standards for a system. In a JTA-compliant system, a mandated standard in JTA is intended to be implemented only by systems that have a need for the information technology services specified by that standard. This means both that the service area is one that is required by the program and also that the guidance in the JTA for applicability of the specific standard indicates that it is appropriate for the program's needs. Additional standards (outside of JTA) may be used to meet requirements, but only if they are not in conflict with mandated standards in the JTA.

Implementing JTA on a Program

To implement JTA on a program, the first step is to develop a JTA profile for the system. This will provide the information that is needed either to assess JTA compliance of an existing program or to plan for JTA compliance in a developing program. A simple process for developing a JTA profile is suggested here, but other approaches could be followed:

1. Create a table from the List of Mandated and Emerging Standards (LMES) (called Appendix B in earlier versions of JTA). Include all standards from the JTA core sections, and all standards from any relevant service areas in domain and subdomain annexes. It is important to check all annexes for relevant service areas, even in domains to which the system does not belong.

JTA Section	Currently Mandated Standard	Applicable ?	Comply ?	Alternate Standard	Comments
2.2.2.1.4.1 Document Interchange	ISO 8879:1986, SGML (with Amendment 1 and Technical Corrigenda 1 and 2)	Y	Y		
	HTML 4.01 Specification	Y	Y		
	XML 1.0	Y	N	Proprietary format	Transitioning to XML in upgrade
2.2.2.1.4.2 Graphics Data Interchange	JPEG File Interchange Format (JFIF), Version 1.02	N			
	PNG Specification, Version 1.0	N			
	GIF, Version 89a	N			

Table 2: *Example Joint Technical Architecture Standards Profile Entries*

2. For each service area, determine whether the service area is applicable to the system.
3. For each applicable service area, identify the standards that are appropriate to the system's needs, using the standard-specific guidance in the JTA. (Note that a standard classified as emerging should not be used if an appropriate mandated standard is available.) Then determine whether the system is/will be compliant with the standards identified.
4. If not, then determine migration plans or justification for non-compliance.

An excerpt from a JTA profile is shown in Table 2 .

The JTA standards profile can be used as a starting point in cases such as these:

- To familiarize designers of a system with relevant standards before design decisions are made.
- To use JTA standards as references for implementers as the system is being developed.
- To develop compliance criteria for testing to ensure that the relevant JTA standards are implemented on the program.
- To establish customers' acceptance criteria.
- To generate migration plans showing JTA standards that will be implemented in later releases of a system, or creating waiver requests if a particular standard cannot be implemented on a system even in the future.

For new programs and changes to existing programs, JTA compliance, and DII COE compliance if applicable, must be in Requests for Proposal and in all relevant contractual documents. The DoD JTA User Guide and Component JTA Management Plan [15] should provide some help with contractual language.

Conclusions

Each DoD Component is responsible for JTA implementation within the Component. Each has unique policies, and additional funding for JTA compliance is often not provided. The OSD direction is clear – JTA is essential to meeting the future requirements for interoperable systems. Getting to this vision of interoperability will be a long-term effort, since JTA compliance is only mandated for new systems and those being upgraded. It is important to realize also that compliance with JTA by itself will not guarantee interoperability between systems. Common data, selection of common options, and sometimes common software, such as the DII COE, will also be necessary to achieve true interoperability. There are likely to be growing pains in the interim, but the overall goal is vital for the future of our military. ◆

References

1. Joint Technical Architecture Version 4.0, Department of Defense, 2 April 2001.
2. Terms and Definitions, DoD Open Systems Joint Task Force (OS-JTF), <www.acq.osd.mil/osjtf/html/approach_terms.html>, 23 April 1999.
3. Joint Technical Architecture Version 1.0, Department of Defense, 22 Aug. 1996.
4. Kaminski, Paul G. and Emmett Paige. Implementation of the DoD Joint Technical Architecture, 22 Aug. 1996.
5. Joint Technical Architecture Version 2.0, Department of Defense, 26 May 1998.
6. Buchholz, Douglas D., Jacques S. Gansler, and Arthur L. Money. DoD Joint Technical Architecture (JTA) Version 2.0, 30 Nov. 1998.
7. Joint Technical Architecture Version 3.0, Department of Defense, 15 Nov. 1999.
8. Gansler, Jacques S., Arthur L. Money, and John L. Woodward Jr. DoD Joint Technical Architecture (JTA) Version 3.0, 29 Nov. 1999.
9. Joint Technical Architecture Version 3.1, Department of Defense, 31 March 2000.
10. CJCSI 6212.01B: Interoperability and Supportability of National Security Systems, and Information Technology Systems, Chairman of the Joint Chiefs

of Staff, 8 May 2000.
11. DoD Regulation 5000.2-R: Mandatory Procedures for Major Defense Acquisition Programs and Major Automated Information System Acquisition Programs, Department of Defense, June 2001.
12. Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR) Architecture Framework Version 2.0, Department of Defense, 18 Dec. 1997.
13. DoD Technical Reference Model Version 2.0, Department of Defense, 9 April 2001.
14. Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification (I&RTS) Version 4.1, Department of Defense, 3 Oct. 2000.
15. JTA User Guide and Component JTA Management Plan Version 1.0, Draft, Department of Defense, 2001.

Notes

1. The term "DoD Components," as defined in DoD Regulation 5000.2-R [11], refers collectively to "the Office of the Secretary of Defense, the Military Departments, the Chairman of the Joint Chiefs of Staff, the Combatant Commands, the Defense Agencies, and DoD Field Activities."
2. The documents identified in JTA Version 4.0 Section 1.6.2 Combined and Coalition Standardization and/or Interoperability are the following:
   - Department of Defense, Directive 2010.6: *Standardization and Interoperability of Weapons Systems and Equipment Within the North Atlantic Treaty Organization*, 5 March1980.
   - Chairman of the Joint Chiefs of Staff, CJCSI 2700.01: *International Military Rationaslization, Standardization, and Interoperability Between the United States and Its Allies and Other Friendly Nations,* 30 Jan. 1995.
   - North Atlantic Treaty Organization (NATO), *Consultation, Command and Control (C3) Technical Architecture (TA) (NC3TA),* 15 Dec. 2000.
   - Allied Communications Publication (ACP) 140, *Combined Interoperability Technical Architecture (CITA)*, 3 May 1999.
3. The DII COE Web site contains such information about DII COE as current implementation status, requirements for changes, future plans, meeting dates for the oversight group and working groups, and links to other relevant Web sites. Information about DII COE changes regularly, since it involves releases of software that may be updated. For current information, check the DII COE Web site: <http://diicoe.disa.mil/coe>.
4. The DoD JTA Web site contains a great deal of information about JTA, including previous and current versions of the JTA document, recent news regarding JTA, and information on how to participate in the JTA development process. The Web site also contains a list of all the organizations participating in the JTA Development Group, with contact info for the representatives from each DoD Component. Following are URLs for the DoD JTA Web site and the JTA Web sites of the Military Services:
   - DoD JTA: <www.jta.itsi.disa.mil>.
   - USAF JTA: <www.afca.scott.af.mil/jta-af>.
   - USA JTA: <http://arch-odisc4.army.mil>.
   - USN JTA: <www.acq-ref.navy.mil>.

About the Author

Judy Kerner is a senior project leader at The Aerospace Corporation in El Segundo, Calif., where she leads activities for the Department of Defense Joint Technical Architecture and related initiatives. Kerner has more than 25 years of experience in software architecture, software engineering, standards, and open systems. She has worked for TRW, Norden Systems, and previously for several commercial organizations. Her assignments have included project management and responsibilities in all phases of the software life cycle, as well as research. Kerner holds a master's degree in computer science from the Polytechnic Institute of New York.

The Aerospace Corporation
P.O. Box 92957
Los Angeles, CA 90009
Phone: (310) 336-6555
Fax: (310) 336-8266
E-mail: judy.kerner@aero.org

The DII COE: Basic Principles and Future Challenges

Doug Gardner
Defense Information Systems Agency

While not a silver bullet, the Defense Information Infrastructure (DII) Common Operating Environment (COE) is a measured, pragmatic approach to software development and integration that is tailored to the Department of Defense. The four basic principles of COE – interoperability, security, customer focus, and best value – are what drives the COE as it responds to past successes and seeks to embrace evolving open frameworks and object technology. This article entails a description of how these principles are addressed in the COE development process. This is critical to understanding the goals of the COE environment: How it is used. How it should be used. Also discussed are the challenges facing the COE as a result of decreasing commercial product life cycles, rising customer expectations, and increased demand to support new communities.

The Defense Information Infrastructure (DII) Common Operating Environment (COE) was created to provide the Department of Defense (DoD) with software processes and products that accommodate the unique corporate characteristics of the DoD. Since its inception, the COE has been embraced by all the major service Command and Control systems and the intelligence community as the basis for combined joint interoperability from the National Command Authority through the commanders-in-chief to the joint task force.

As embodied in the Global Command and Control System (GCCS) and its service variants, COE has been employed in all major theater operations. These include the U.S. Central Command operations SOUTHERN WATCH (no fly-zone enforcement) and DETERMINED RESPONSE (USS COLE aftermath), U.S. European Command operations DELIBERATE FORGE (NATO Air Operations), JOINT FORGE (Stabilization Forces), and SILENT PROMISE (South African relief).

As COE embraces evolving open frameworks and object technology, such as the Web, publish-and-subscribe data sources, portals, and component software, hundreds of new system integrators have moved to the COE.

What drives the COE as it responds to these past successes and seeks to support a broader application base? The answer to this question is encompassed in the COE's four basic principles: *interoperability*, *security, customer focus*, and *best value*. A description of how these principles are addressed in the COE development process is critical to understanding the goals of the COE environment, how it is built, and how it should be used.

Interoperability

Interoperability of joint systems is critical to success on the modern battlefield. However, interoperability is about making limiting choices. The DoD has tried many initiatives to make joint systems interoperable, most of them based on developing and mandating standards (such as Ada, POSIX[1], and the Joint Technical Architecture). While these efforts have had some success, the COE goes beyond interoperability through standards and provides interoperability through products. Common products make for common software behavior and reduce the number of avenues that developers can use to move away from a common, interoperable implementation.

> *"The COE, as the primary vehicle for providing controlled infusion of new technology across a large number of systems, is squarely in the crosshairs of the expectations gap."*

The DII COE is based on the thesis that having the exact same software on both sides of an interface is the most effective way to ensure consistent behavior across the interface. Thus interoperability between systems can be measured by how much of the exact same software is shared between the systems. Interoperability in the COE is a spectrum: its minimal level of interoperability being federation (the ability to run two applications on the same platform without stepping on each other), and the maximum level of interoperability being two systems that are entirely identical except for domain or functionally specific mission applications.

There are some practical advantages to high levels of interoperability as defined by the COE, some of which are particularly important in the DoD environment. For example, we expect that in future contingencies a military user will need to run a functionally specific application on a system supplied by another service (e.g., the Army logistician who needs to run an Army application while assigned aboard a Navy ship as part of a joint task force). High levels of interoperability, as defined by the COE, will make this easier by ensuring that the underlying infrastructure of both the originating Army system and the receiving Navy system use the same software.

It is a continual challenge for the COE to balance the need to build common software with the legitimate requirements of system developers for specialized or unique services. In each of these cases, the potential for providing services in the COE that could reduce interoperability (for example, multiple products that provide the same service) must be weighed against the specialized requirement. This approach is designed to help the community make limiting choices together, and while it has resulted in a substantial amount of community agreement, it is an approach that is not very popular with developers and users who are accustomed to having complete control of functional and system design decisions.

A common statement from a program exploring using the COE is, "I can't use the COE because it doesn't have or do _____ (fill in the blank)." If the COE can be expanded to handle that blank space, then every effort is made to address the issue in the COE software baseline. If the request is incompatible or would introduce new opportunities for COE customers to make conflicting decisions, then the requested capability is not

included in the COE. If the program then chooses to not use the COE, the program should clearly understand that they have just prioritized their missing desire above joint interoperability, or at least the level of joint interoperability you get from the COE. In some cases, this is an understandable trade-off for a program to make (their need really is more important than the gains afforded by the COE). However, in most cases, the decision to not use the COE will make the integration of that program's software into a joint system considerably more difficult.

Security

Security is an important factor in the COE, and as might be expected, has been growing in importance over the past few years. Since the COE is not a system, it does not go through the formal security accreditation required of DoD systems. Instead, components being included in the COE are assessed against a stringent set of security guidelines, and are occasionally rejected if their acceptance would create an unacceptable level of risk in end systems. The COE provides recommendations, in the form of default software configuration settings, for how COE customer systems should use COE components to maximize security. Ultimately, however, and this is a very important point, the security posture and level of acceptable risk are the decision of the developers of the systems that use the COE.

Aside from the secure components and configuration guidance, the COE provides additional security-related services to developers of COE-based systems and applications. First, by using the COE, a customer system inherits a basic amount of security that serves as a community-wide, common level of reasonable basic security. This basic security serves as a starting point for developers and allows them to increase the security of their systems to meet specialized security requirements.

The second primary security service provided as part of the COE process is a watchdog organization (the COE Security Team) that continuously monitors threats to COE software, informs the community of potential concerns, and ensures that appropriate patches or configuration advice are made available as quickly as possible. The COE Security Team monitors DoD security initiatives (such as Information Alert Vulnerability Assessments), as well as security publications and commercial product announcements, to identify security issues that are relevant to the COE customer community. This focus on security issues associated with COE components is designed to reduce the amount of vulnerability research required by COE-based system developers and administrators.

A third focus of the COE security support effort is to provide tools that system integrators and application developers can use to enhance the security of their products. One tool is a configuration-based tool that examines a machine and identifies potential security vulnerabilities such as file permissions on UNIX that are too permissive or scripting conventions that can be readily exploited. Another tool is a set of interfaces that allow application and system developers to embed secure authentication and data transport into their software. These tools can be used by developers to build security into systems, and they will make it easier to build systems with consistent security implementations. The tools will be expanded over time to support new security mechanisms and services.

"This component-based view of system development encourages system integrators to meet system requirements by looking for existing components ..."

Customer Focus

Ultimately, the DII COE is about building systems. In the DoD "system-building universe," there are four primary COE customers: system integrators, application developers, system administrators, and functional users. This diverse set of customers has very different requirements. The COE's success depends on our ability to balance the capabilities provided in the COE across these four groups.

System Integrators

System integrators are those DoD development organizations that are responsible for providing integrated, full-service system solutions that include both general purpose capabilities (such as office automation and Web browsers), as well as functionally specific capabilities (such as situational awareness, transportation management, or financial services). For the military command and control community, the primary systems that use the COE are the Global Command and Control System (GCCS), GCCS-Maritime, GCCS-Army, Theater Battle Management Core Systems, and the Army Battle Command System.

System integrators rely on the DII COE to provide a common integration methodology that supports widespread sharing of applications across systems and ensures that applications developed according to COE guidelines will peacefully coexist in an integrated system. The COE allows system integrators to view the integrated system as a series of components that can be packaged together or flexibly deployed to meet specific needs of system users.

This component-based view of system development encourages system integrators to meet system requirements by looking for existing components, either provided by the COE or built by another COE-compliant developer in the DoD. This leveraging of other development efforts allows the system integrator to focus resources on the new or functionally unique portions of the system rather than on general purpose tools and components.

The COE has two key rules designed specifically to support system integrators: software services provided by the COE are not retired except through a process of community agreement, and services in the COE are upgraded with an eye toward full backward compatibility. This ensures that applications built using the old services continue to run in the upgraded environment without any modification. These goals allow system integrators to use COE services or COE-based applications knowing that services in the COE will not disappear without warning, and that all customers will have a say in the retirement schedule. For a large system that includes hundreds of functional components, these guarantees allow the system integrator considerable flexibility in upgrading individual components of the system over time, instead of having to reengineer all capabilities before any new infrastructure features can be fielded.

System integrators can choose to include in their systems software developed by the COE. The COE provides the standard implementation for key command and control functionality, specifically the data management, dissemination,

and visualization components that make up the Common Operational Picture (COP). The COP is a framework and a set of standard capabilities built against that framework that provide a common set of data, an integrated display capability, and a robust data replication mechanism in support of situational awareness across the joint community. Also, the COE provides a set of government-built services that are not supported by commercial products (such as profiles, which extend a user's account definition to support shift work and multiple duties, and cross-platform account management, which includes a single tool for creating accounts that work across all COE-supported platforms).

Application Developers

Application developers are those DoD development organizations that build software to address a particular functional area such as weather, intelligence, or logistics. While applications are usually built to be part of a particular system, some are built to be fielded as components of multiple systems. Two current examples of joint mission applications are the Integrated Imagery and Intelligence application suite and the Air Tasking Order Exchange capability.

Primarily, the COE supports application developers by allowing them to focus their resources on developing mission-specific functionality instead of infrastructure. Before the acceptance of the COE, functional applications were usually not built to be part of a larger system. Each application required its own account management tools, map package, data management engine, etc. For many applications, these components could eat up well over half of the development budget without ever providing the end user any specific functionality. By using the COE, the money allocated to develop infrastructure components can instead be reprogrammed toward meeting more of the end user's functional requirements.

In addition, reliance on COE infrastructure components also helps the application developer ensure that the application can be integrated into multiple systems with minimal effort. For example, community acceptance of a standard mapping package means that separate versions of the application for different map engines are not required. The application developer has a clear idea of the target environment during development and therefore can make appropriate decisions without a great deal of coordination.

The COE also provides application developers with a public clearinghouse for requirement and product information. The DII COE has chartered 19 technically focused working groups to document requirements, incorporate technical advice, and recommend products for inclusion in the COE. These working groups conduct a significant amount of research, both in terms of what is commercially available and how well various products fulfill community requirements. These are public forums that include experts from across the DoD community. Participation in these groups, or at least monitoring their progress, is a good way to reduce the amount of resources that a development organization has to spend to research both industry and the rest of the DoD.

" ... the COE supports application developers by allowing them to focus their resources on developing mission-specific functionality instead of infrastructure. "

System Administrators

System administrators are the people tasked with making DoD systems run in the field. They are usually military specialists and are almost never the same people who develop the applications or integrate the systems to be operated and maintained. To do their job properly, they need clear system documentation, automated installation and upgrade tools, and predictable system environments.

The COE supports system administrators by forcing application developers and system integrators to consider, as part of the development process, the impact of design decisions on the workforce that will make the systems run in the field. As part of the overall COE philosophy, it attempts to minimize the effort required to maintain and upgrade COE-based systems while providing enough flexibility to allow operational sites to adequately control their local information technology resources (e.g., to install the software they need to accomplish their mission).

The COE tool supporting these goals is the COE Installer, which provides a point-and-click interface for loading, updating, and configuring software on COE-based systems. The COE Installer performs specialty features like installations from a network server, dependency checking to ensure required components are loaded in the proper order, and version checking to ensure that applications have access to the proper versions of commercial and government-built infrastructure components. COE rules about separate directories for each component ensure a predictable "laydown" of software on each machine, and that the installation of a new component will not remove or overwrite software required by an already loaded component.

Functional Users

Functional users are the military operators who use the systems built on the COE. They use their computer systems to perform practically all aspects of their day-to-day duties, as well as to prosecute joint military operations. Systems built on the COE support users across the spectrum of military operations from routine administrative users to those conducting operational planning and execution. COE-based systems also provide support to users at all echelons of the command structure hierarchy (from the National Command Authority to the foxhole).

The functional users are the ultimate customers of the COE, but what the COE provides is largely invisible to them. Although concepts such as component installation, system integration, and software resource management are outside the functional users' scope (and interest), these concepts provide the processes and software framework for joint cooperation and integration that make it possible to execute modern joint operations. The COE provides the mechanism to bring together applications from a variety of systems, quickly and under less than ideal conditions, to automate a joint task force. One immediate pay off of the COE in this area is that it has almost completely ended the practice of each new functional capability being delivered to the end user as a new system with new hardware, training, and administration requirements. New functionality is now commonly delivered into a system, so users do not have to manage multiple workstations each providing only a portion of the necessary information in order to do their jobs. Systems like GCCS, and their service counterparts, now serve as the host systems for new functionality

being offered to users.

Another key benefit to the functional users is that, for the same amount of funding across the DoD, the COE allows less to be spent building duplicative (and often non-interoperable) infrastructure components. This frees up more money to be focused on the functional requirements that directly support mission accomplishment. This redirection of resources is already starting to become evident in some functional domains within the DoD and is likely to become more obvious over the next few years as current service and joint systems upgrade their infrastructure to the latest version of the COE.

In the area of the COP, the COE provides functional users a situational awareness capability that can expand and change as new sources, sensors, and decision tools become available. The challenge from the COE perspective is to make dozens of COP applications and decision tools, built by a wide variety of government and commercial organizations, appear to the end user as if they were built as part of a single, integrated suite of applications. By providing just such a framework, the COP has become the primary integration mechanism for command, control, and intelligence data across the DoD. It will continue to evolve to allow the broadest amount of flexibility for contributing applications to bring additional information to the common display and to facilitate the secure distribution of data needed by decision-makers at all levels.

Best Value

Although not the primary focus of the COE, an extremely important byproduct of achieving system and application interoperability through common software is cost savings. The key aspect of cost savings in the COE is the use of commercial products, which make up about 85 percent of the COE. In almost every case, COTS products are less expensive for the government to acquire, modify, and enhance than government-built components.

As for the government-built products in the COE, they are developed either because available COTS products would make the COE less interoperable or because the required functionality does not exist in any commercial product. Although it seems axiomatic that using an existing government-built product rather than building your own would save money, the amount of money saved has been difficult to quantify. This is probably because most programs don't track cost savings, but the fact is that the DoD has not invested the time and resources required to quantify costs attributable to the COE.

However, there are areas where savings due to community-wide adoption of the COE can be quantified. A particularly good example is the Integrated Imagery and Intelligence set of mission application capabilities built by the Navy and being fielded on each of the COE-based major service command and control systems. Instead of four different development efforts, the DoD will pay for only one. As this model is applied to each of dozens of functional areas across the military, this aspect of cost savings will likely represent the biggest financial advantage of the COE for the DoD.

"The COE, as the primary vehicle for providing controlled infusion of new technology across a large number of systems, is squarely in the crosshairs of the expectations gap."

However, there are packaging, and at higher levels of COE compliance, reengineering costs associated with migrating to the COE. The initial cost to move to minimal COE compliance is usually very low, partly because the COE was designed that way and partly because minimal COE compliance is based on adhering to generally recognized good software development practices. Depending on how tightly an application is integrated with its infrastructure, achieving higher levels of COE compliance can incur moderate to high costs. The savings in long-term maintenance balance some of these migration costs, but mostly the costs for migration to the COE should be viewed as the cost of becoming joint.

One other cost-saving outcome of the COE's role as a repository for common capabilities is that the COE has become a single forum that represents service and agency software requirements to industry. This allows the COE (usually through technical working and advisory groups) to represent a broad range of the DoD in discussions with industry. It also allows the COE to arrange COE-wide licenses for certain key capabilities, such as printing and Web services.

Future Challenges

The COE faces many challenges in the future, particularly in the areas of keeping up with technology, maintaining a collaborative atmosphere with our customers, balancing the use of commercial products and services with the need to maintain open software solutions, and expanding the COE to take advantage of other services and technologies.

The "Expectations Gap;" the Treadmill Keeps Getting Faster

The most significant challenge for DoD software development in general and the DII COE in particular is the growing mismatch between the amount of time it takes to field a system and how quickly commercial industry is moving. The COE initially assumed service systems would upgrade their software and hardware infrastructure every three years. The reality, however, is that the current versions of fielded systems will not be upgraded for five to seven years, for a variety of reasons: specialized security requirements, in-depth functional testing, expense of retraining users and system administrators, operational concepts that lag technological innovation, and scheduling availability for operationally deployed forces, just to name a few.

So what DoD software developers face is a growing expectations gap by users who see new capabilities in the commercial marketplace that are still years away from being systematically deployed in the DoD. The COE, as the primary vehicle for providing controlled infusion of new technology across a large number of systems, is squarely in the crosshairs of the expectations gap. We need to move fast enough to keep up with the lightning pace of industry while not leaving any legacy systems or applications behind.

A factor that will increase the expectations gap is the frequency with which commercial products are being replaced and/or retired by their manufacturers. New and improved products are being produced much faster than large-scale system developers can keep up. With each new release, the commercial business mind-set is that an older product becomes *unsupported* (both to save the manufacturer on the number of baselines to maintain and to *encourage* customers who haven't upgraded in a while to move to the later

version). For some commercial products, the release-to-retirement cycle fits inside the typical DoD system "develop, test, and field cycle." That means that in the time between code freeze for validation, certification, training, and fielding, some of the commercial products in the *frozen* baseline are becoming unsupported by vendors. While the COE guarantees that its government-built interfaces will remain backward compatible and fully supported, it is not possible to make the same claim for commercial products.

This Only Works if Everyone Works Together

The DoD consists of hundreds of autonomous, decentralized software development and acquisition organizations, each of which contributes a portion of the overall capability required to prosecute joint operations. With so many agendas and specific needs that ultimately are required to come together to support the decisions of a single commander, it is critical that there be an open dialogue to reconcile the conflicting demands of systems contributing to joint operations. The COE provides a forum for discussing the technical tradeoffs associated with joint software development. It can continue to be a useful part of the overall joint solution if the services and agencies that participate continue to make being joint a priority.

The Not-So-Hidden Threat From Industry

The DoD is a highly competitive arena, at times within the government but certainly among the defense contractors and commercial vendors who provide government software services.

From the standpoint of the commercial marketplace, the COE conflicts with the corporate agendas of most commercial vendors. The common development and integration approach of the COE discourages any program to become dependent on a proprietary approach offered by a particular vendor. The COE enforces an anti-monopoly stance by providing services in such areas as cross-platform support and by including multiple commercial products where doing so allows systems to make cost/feature tradeoffs without sacrificing interoperability.

A related challenge for the DII COE is trying to balance the DoD's desire to use commercial products with corporate business models that push for product uniqueness and proprietary approaches. A perfect example of this is the DoD goal of cross-platform consistency. The ideal is that platforms in the DoD should be interchangeable, that is, they should provide a common set of services invoked the same way. This would allow applications and software tools to be more readily shared across systems, thus saving the DoD millions (perhaps billions) of dollars and ensuring consistent behavior for all users. Instead, commercial industry spends billions to ensure that the ideal is never reached – there is no business case for making a product the same as a competitor's. Even where standards exist, such as Structured Query Language for relational databases, the *extensions* provided by each database vendor ensure that applications built for one product cannot be moved to another product without significant reengineering.

Controlled Growth for the COE

There is considerable pressure on the COE to expand to support new technical requirements, particularly in the areas of real time and tactical systems support.

Expansion into the real-time environment will require support for a much larger set of hardware and operating system configurations. It will also require fundamental reengineering of some COE applications to both adhere to more stringent processing requirements and to take advantage of new services provided by real-time operating systems. The differences in system development and integration philosophies between the real-time and non-real-time communities have already challenged some of the core COE concepts. There is a clear need for the COE to provide the tools and products that allow integration between the decision-making systems that support a joint task force and their real-time counterparts. This will be a significant focus area in future deliveries of the COE.

Providing support to the tactical community will challenge the COE to operate on smaller hardware and to support future developments in wireless technologies, hand-held Personal Digital Assistants, and radios. More flexibility in the amount of bandwidth used, more control over the flow of data, and more visualization options will be required. The requirements in this space are just beginning to be defined, but this is also clearly an area that the COE will need to support in future deliveries.

The challenge to the COE will be to incorporate these capabilities while maintaining a stable baseline for current COE customer systems that are in the field.

Conclusion

The DII COE is a measured, pragmatic approach to software development and integration that is tailored to the DoD. It is a customer-driven and cost-conscious process that results in products that are interoperable and secure. The challenges facing the COE over the next few years are significant as the trends of decreasing commercial product life cycles, rising customer expectations, and increased demand to support new communities converge. As the COE evolves in the future, it is critical to understand that it is not a silver bullet. Successful software development in the DoD still requires good systems engineering, disciplined development processes, and detailed coordination among related applications and systems.◆

Note

1. POSIX is a registered trademark of The Institute of Electrical and Electronic Engineers, Inc. (IEEE).

About the Author

Doug Gardner has worked on practically every part of the Defense Information Infrastructure (DII) Common Operating Environment (COE) since he began with the Defense Information Systems Agency (DISA) in 1996. After serving as the Common Support Applications Team chief for two years, he was named as the COE chief engineer in April 2001. He has a master's degree in defense policy from The Claremont Graduate School and a bachelor's degree in electrical engineering and computer science from Rice University. Prior to coming to DISA, he worked for the Jet Propulsion Laboratory as a software developer on a variety of command and control, intelligence, and modeling and simulation systems. Gardner is also a major in the U.S. Army Reserves.

Defense Information Systems Agency
Phone: (703) 681-2328
E-mail: gardnerd@ncr.disa.mil

The DII COE: An Enterprise Framework

Dr. Gregory Frazier
SAIC

The Defense Information Infrastructure (DII) Common Operating Environment (COE) is a framework for the creation of a set of cooperating computing enterprises. Its goals include the elimination of "stove-pipe" systems and cost reduction via software reuse, reduced need for system administration, and simplified system integration. This article describes in brief the salient features of the DII COE. It describes some of the challenges related to migrating systems to a DII-compliant environment, and concludes by arguing that the key to the successful use of the DII COE is for all participants to be aware of and work toward building and maintaining an extensible, general-purpose environment.

In 1994, the Defense Information Systems Agency (DISA) began development of the Global Command and Control System (GCCS). The goal of GCCS was to link the commander-in-chief (CINC) sites into a single planning and logistics network. It utilized an architecture that was developed in the Navy Joint Maritime Command Information System (JMCIS) program for integrating software modules developed by multiple contractors. In 1995, GCCS was operational and DISA had begun work on the Defense Information Infrastructure (DII). In 1998, GCCS version 2.0 was the first Department of Defense (DoD) system to utilize the DII Common Operating Environment (COE) as the basis for its implementation, being deployed on top of the DII COE Version 3.1.

Since then, the DII COE has been or is being used as the framework for more than 100 DoD computing systems (see Figure 1). The term "COE" has moved into common parlance for corporate Chief Information Officers, and foreign governments are looking to construct common operating environments for their computing enterprises. Although the DII COE has been in existence for more than three years and has enjoyed remarkable success, it is not widely understood. This article offers insight into the DII COE architecture and the rational for its design.

An Enterprise Focus

The DII COE is a framework for the development of enterprise computing systems. In order to understand the decisions made regarding its design, it is important to first understand what an enterprise computing system is, and what it means to be a framework.

An enterprise is a venture – the act of an organization working toward a common goal. Enterprise computing is the establishment and use of a computing system that supports this venture by implementing the business processes of an organization. The DoD is a grand-scale organization with approximately 11 million military and civilian participants. Its purpose is equally grand: the defense of the United States of America. This is, of course, too large an organization and too complex a venture to be understood as a single enterprise. Thus the DoD is partitioned into many smaller organizations, each executing their own enterprise. However, these are collaborative enterprises – the organizations work together to support common goals. The DoD enterprise computing system is a system of systems in which information and services are shared across enterprise boundaries.

A framework is a software package that supports the creation of architecture by guiding developers toward a particular set of design patterns. If the developers conform to the framework and utilize its services, then the resulting system will reflect the desired architecture. It is important to note that a framework is not itself a system. The DII COE is a framework for the construction of modular, scalable, distributed Command, Control, Computer, Communications, Intelligence, Surveillance and Reconnaissance (C4ISR) computer systems. It is a collection of tools for the creation of these systems; it is a set of software modules that can be (re-)used to construct these systems. Or, to quote the DII COE Integration and Runtime Specification [1]:

> "The DII COE emphasizes both software reuse and data reuse, and interoperability for both data and software. But its principles are more far-reaching and innovative. The COE concept encompasses:
> - An architecture and approach for building interoperable systems.
> - A minimal but extensible security architecture and a set of security services.
> - An environment for sharing data between applications and systems.
> - An infrastructure for supporting mission-area applications.
> - A rigorous definition of the runtime execution environment.
> - A reference implementation on which systems can be built.
> - A collection of reusable software components and data.
> - A rigorous set of requirements for achieving DII compliance.

Figure 1: *Military Systems Operational or Being Developed Using the DII COE*[1]

Army

- AALPS
- ACGS
- AMBISS
- AMDWS
- AMDPCS
- AMPS
- ANGSC-52
- ASAS
- ASD
- ATCS
- ATLAS
- BCTP
- BMC3
- BSM
- CINC CSA
- CNCMS
- CNPS
- CR/HMS
- CSCE
- CSSCS
- CTIS
- C4IJM
- DCARS
- DTSS
- FATDS
- FAAD C21
- FIRESTORM
- GCCS-A
- GCSS-A
- IBDAS
- IMETS
- ISYSCON
- LW
- MCCCC
- MCS
- MFCS
- PEGEM
- RCAS
- SAS
- TAIS
- TCAIMS
- THAADBMC3I
- TPSOPS
- TSIU
- UAV
- WARSIM

Navy

- AADC
- CADRT
- COMDAC
- CCS
- CUB
- CV/TSC
- GCCS-M
- IUSS
- JMPS UPCs
- KSQ-1
- LAMPS
- MEDAL
- METOC
- MPA
- MPAS
- MSBL
- NAVSSI
- NFCS
- NSPF
- NSS
- NSSN
- PTW
- REDS
- SFMPL
- SH60 MPS
- SRMT
- SCCS
- TACLOGS
- TAU
- TEAMS
- TERPES
- TCAC
- TDSS
- TTWCS
- VTC

Air Force

- AFWeather
- AMC BDM
- AMS
- AWACS
- A2IPB
- CSEL
- DCAPES
- Defense IEMATS
- FORTE
- GBS
- GCCS-AF AETC
- GCSS-AF IF
- IMDS
- IMOM
- ISC2S
- MAMS
- Rosetta
- SBMCS
- STRATCAT
- TBMCS

Marine Corps

- AFDI
- ARTDF
- GALE
- GCCS
- GCCS-I3
- GCSS
- JCACTD
- JCALS
- JDISS
- JDP
- JMCSID
- JMPS
- JSCGS
- JTAT
- Joint Tactical Term
- Joint Targeting Tool
- JWARN
- MEPED
- MIDB
- TNP
- JMNS

- An automated tool set for enforcing COE principles and measuring DII compliance.
- An automated process for software integration.
- An approach and methodology for software and data re-use.
- A set of application program interfaces (APIs) for accessing COE components."

Fighting Stovepipe Systems

A key driver for the DII COE is the fact that the DoD is composed of multiple cooperating enterprises. While each of these enterprises is dedicated to accomplishing a specific goal, the data products of one enterprise are consumed by other enterprises as they work together to accomplish the overall goal of defense. However, most computing systems built for the DoD are [historically] stovepipe systems; systems that operate in total isolation from the rest of the computing environment. This is in contrast to a *system of systems* model, where data flows seamlessly from one business process to another.

Stovepipe systems do not support abstract goals such as extensibility or interoperability that are central to the DoD's ability to field cooperating enterprises. A computing architecture that is intended to unify the DoD must be capable of adopting the configuration appropriate for a particular enterprise's mission while maintaining a commonality that will allow that enterprise to interoperate with other DoD computing enterprises.

The DII COE is, first and foremost, a framework intended to prevent the creation of stovepipe systems and promote cooperative enterprises.

Other DII COE Goals

While the principal goal of the DII COE is to eliminate stovepipe systems, there are a number of important drivers for the design of the DII COE architecture:

- Increased software reuse. While each enterprise within the DoD has a unique mission and a computing system to support that mission, there is inevitably some functionality that the enterprise will have in common with others.
- Reduced need for computer system administrators. The DoD personnel using computer systems should be warfighters whose mission is supported by the computing system, not system administrators whose mission is to support the computing system.
- Improved technology insertion. The enterprises that comprise the DoD are long-lived enterprises. Their life span dwarfs the technology cycle, and thus there must be an avenue for technology insertion in DII COE's computing architecture.
- Simplified system integration. Given the distributed development environment in which multiple contractors contribute modules to one or more integrators, and each integrator delivers the resulting system to multiple organizations to field, it becomes important to push as much integration responsibility as possible to the developers and the deployers.

The following section will describe the DII COE and discuss how it achieves these goals.

The DII COE Framework

The first DII concept dealt with by a system integrator, an administrator, a developer, or even as a user is the segment. A segment is a unit of software or data that has been packaged such that it can be installed on a DII-compliant computer using the software installation tool of the DII COE. All software that is to be installed on a DII computer must first be put into segment format.

What is typically done for government off-the-shelf software (GOTS) is that software developers segment it before delivery to a program engineering office. For commercial off-the-shelf (COTS) software, the program office typically purchases the product and then hires contractors to segment it. However, the model envisioned for the DII COE was for commercial vendors to put their software into segment format.

Either way, an engineering office receives software in segment format, accompanied by a set of documents that include installation procedures, a users manual, test plans and test results, version description, and specification of the segment's compliance level. The compliance level is the degree with which the segmented software is in compliance with the DII Integration and Runtime Specification (I&RTS – discussed in the next section). The engineering office tests the segment, and then makes that segment part of its build list. It is now available to be delivered to the field, either as part of a standard build or as an optional segment.

Segmentation is controversial for several reasons. First, it is an additional cost. Program offices dislike having to spend the money to have software segmented on top of the cost of developing or purchasing said software. This is particularly resented with COTS software, which generally arrives in an installation package. Second, there are not extensive tutorials on segmentation, and the software used to support segmentation can be a bit cranky. The learning curve can be very steep for newcomers to segmentation. There are, however, a number of reasons to utilize a single software-packaging standard for the DII:

- Segments facilitate configuration management. All software in segment format shares a standard versioning system, a standard naming system, and has a standard set of documents associated with it. With a single program (the COEInstaller) you can see which segments are installed on a given machine, what resources a segment requires, and what other segments a given segment depends upon.
- The same installation software is used to install every segment on every computer. System administrators are not forced to deal with the vagaries of how different developers decide to have their software installed.
- Segments facilitate software reuse. Segmentation forces the developer to prepare the software for integration. Once in segment format, a software package can easily be transferred between program offices and used in a variety of contexts.

When developing a segment, the most important fact to keep in mind is that a segment should be of general utility. It is possible to segment software in such a way that it is useful only for a very specific purpose. This defeats the reuse goal of segmentation and ensures that the same software will have to be re-segmented. A better approach is to segment software in as general a manner as possible, and then provide a separate configuration segment that modifies the installation for a specific need. This is the approach taken for infrastructure services in the COE such as the iPlanet Enterprise Server and the Oracle Database Management System (DBMS).

The Common Operating Environment

The common operating environment is just that; an operating environment that is common across computing systems and problem domains. There is a common misconception that all DII segments are part of the COE, or that by segmenting an application one is making it part of the COE. This is not the case. Segmenting software makes it available to the DII, but not all software is common. The I&RTS refers to segments that are not part of the

COE as mission applications.

Mission applications are software packages that are intended to provide functionality that corresponds to a specific problem. While it is important for mission applications to comply with the I&RTS and conform to the conventions of the DII COE, these applications will not be widely used and thus not part of the COE.

The software that comprises the DII COE is placed into three categories: the kernel, the common support applications, and the infrastructure services. The kernel is that portion of the DII COE that is installed upon every computer. It is the bootstrap portion of the COE, containing such functionality as the segment installer, the file permission and disk partition configurations, etc. The common support applications are desktop applications that are deemed to be of general use: a word processor, an email client, a Web client, and others. The infrastructure services are services that are deemed to be of general use: various DBMS products, a Web server, a directory server, and others.

Every computer that is to be a *DII computer* must have the kernel installed upon it. The rest of the COE – the segments that comprise the common support applications and the infrastructure services – is available for installation, but is not required to be present on every machine. What is required is that they be available to any system. In other words, a segment developer or system designer can make the assumption that the segments that are in the COE are available for use and can incorporate those segments into his/her design without knowing anything else about the deployment environment.

The Integration and Runtime Specification

The COE and segmentation are specified in the I&RTS; it describes how a computer is configured such that it is DII compliant, and how to build software segments that are DII compliant. It describes the various types of segments, including software segments, COTS segments, data segments, database segments, Web segments and others. In its own words:

> "This document is an engineering specification that describes how modules must interact in the target system. System architects and software developers retain freedom in building the system, but runtime environmental conflicts and data conflicts are identified and resolved through automated tools that enforce COE principles [1]."

It is required reading for anybody who aspires to build, test, or integrate segments, or anybody who wishes simply to understand the DII COE. It is available at <http://dod-ead.mont.disa.mil/cm/geneal.html>.

Compliance Levels

The commonality of the COE rests on the fact that mission applications use the COE to provide common functionality. If segments provide their own implementations of the COE functionality, then the benefits described above are not observed: no cost savings through reuse, no cost savings due to reduced system maintenance, no facilitation of technology insertion, and no interoperability enabled via shared infrastructure services. Instead, we would see a retreat to stovepipe systems delivered as segments.

The purpose of compliance levels is to provide a metric of how well a segment makes use of the COE. This provides a quantitative measure both of how well a segment is integrated with the COE and the progress that a segment makes over successive deliveries. The I&RTS specifies eight levels of compliance. At lower levels, we are dealing with the basic structure of segmented software:

Level 3

Platform Compliance: The application is well behaved in the platform context. It runs on a version of the standard operating system that is supported by the DII. It does not utilize hard-coded port assignments. It does not bypass the standard GUI APIs for the platform. It can safely execute alongside DII-compliant applications, etc.

Level 4

Bootstrap Compliance: The application is packaged in segment format (i.e. it can be installed and deinstalled using the COEInstaller).

Level 5

Minimal DII Compliance: The segment conforms to the I&RTS to the extent that it does not represent a security risk and does not negatively impact system configuration when installed.

In levels 6-8, the compliance levels measure how well the segment makes use of the COE. Appendix B of the I&RTS provides questionnaires that allow developers to evaluate the compliance level of their segments.

DII Adoption Challenges

In this section, barriers to DII adoption are discussed. These barriers are organized in three categories: cost, platform availability and knowledge.

Cost

One of the major obstacles to adoption of the DII is the expense of migrating a given system to use of the DII COE and/or achieving a given compliance level. There are two types of systems that experience significant cost penalties when pursuing DII integration: legacy and leading edge.

Legacy systems experience high costs due to the need to reengineer portions of the system to achieve a given level of DII compliance. This cost can be prohibitive, particularly if it means replacing a constituent product (e.g., replacing the database). When dealing with legacy systems, it is important to understand when to enforce the DII compliance directives and when to relax them. These directives do not make sense for many legacy systems, particularly those that are at the middle or toward the end of their life cycles and will never observe the maintenance cost reductions potentially available from the COE.

Leading-edge systems experience high integration costs due to the lack of support provided by the COE. The DII realizes cost savings via the reuse that segmentation promotes. A system that is utilizing a leading-edge technology may be ahead of the COE and will bear the brunt of the design and segmentation costs without receiving the benefits of reuse. For a small program that anticipated obtaining the majority of its functionality from a COTS product, the relative cost of segmenting that product can be prohibitive.

Worse, a leading-edge system that makes use of a not-yet-adopted technology runs the risk that the DII COE will at some point incorporate the technology but in a way that is incompatible with how the system used it. This risk can be alleviated by active participation in the DII COE Technical Working Group (TWG) process[2] – but the program must then dedicate manpower to track or participate in the TWG and [possibly] experience schedule delays awaiting clear direction from the TWG and/or Architecture Oversight Group (AOG). Also, program management offices that are new to DII may be unaware of the TWGs' existence or not know how to contact their service or agency AOG representative[3]. Leading-edge pro-

grams should be blazing the trail, showing DISA how (or how not) to integrate new technologies. Their participation in the DII COE TWGs must be encouraged and supported, in order to reduce costs both for those programs and for the DII COE.

Platform Availability

The DII COE (Version 4.x) currently supports three platforms: Microsoft Windows, Sun Solaris running on Sun workstations, and Hewlett-Packard HP-UX. All other platforms are unavailable if one is required to be DII compliant. There is a Kernel Platform Compliance (KPC) Program[4], but only two computing systems have successfully gone through the KPC, and both did so on version 3.3 P1 of the DII COE kernel[5]. Since the version 4.0 kernel had an almost entirely different kernel code baseline from the 3.x version, neither platform has been recertified for the current version of the kernel.

Even among the three core COE platforms, the kernel is not the same. The paucity of platforms supporting the DII COE and the variance among the DII COE platforms are the result of the kernel being specified by its source code. To quote the Defense Information Infrastructure (DII) Common Operating Environment (COE) Kernel Platform Compliance (KPC) Program Document for DII COE Kernel Version 4.200 [2], the KPC Program certifies that a platform "executes the Government Supplied Kernel Source code with the same behavior as the current 'Reference Platforms'." In other words, a KPC-certified computer is one that runs the kernel source code that it receives from the DII COE Engineering Office. This has three ramifications.

First, for platforms that cannot share the code base (e.g. Windows NT and Unix/Motif), the kernels differ in unspecified ways. Second, for platforms that desire to support DII, their only recourse is to port each version of the kernel to their platform, which is logistically untenable. Third, there is no mechanism for non-traditional platforms (non-uniform multi-computers, for example) to achieve DII certification via the KPC Program. While the KPC Program does include a test suite to evaluate the kernel port, this test suite does not comprise a specification of the kernel. It is the reference implementation source code that acts as the specification[6], so any platform that cannot compile that source code cannot be in the DII.

Getting the Word Out

The final and most significant impediment to DII COE adoption is the general lack of knowledge regarding what the DII is, what the COE is, and how a program can leverage the DII to facilitate the development of their system. The cost and platform support issues discussed in the previous sections can be addressed by the thoughtful requesting and granting of waivers to allow systems to conform to the intent of the I&RTS without being hamstrung by its rules. To do this, however, requires that both the program management office and the contractors executing the contract have a thorough understanding of the DII.

While there is a great deal of data available regarding DII and the DII COE, it can be challenging for newcomers to discover the information that they need. The principal source of information is the I&RTS. This is required reading for anybody who intends to develop an application for use in DII, integrate a DII system, manage a DII-related project, or deploy a DII system. However, the I&RTS is not a how-to manual or a tutorial – it is a specification of the DII runtime. The DISA and DII COE Web sites also provide a great deal of information. It is incumbent upon the DII participant to find the information that they require from these sources.

Conclusion

The ongoing success of the DII and its COE depends upon a number of factors. First, the DII must maintain its relevance. This means continuing to support technology insertion efforts. It also means that the organizations that are deploying DII-compliant computing systems must understand not only the letter of the I&RTS, but also its intent. They must execute their system development such that the result fits with the spirit of the DII. This can be captured in words such as *extensible*, *open*, and *reusable*. Wherever possible, avoid one-of-kind solutions.

If a program requires a common service established with a particular configuration, deliver two segments: one that contains the common service, and a separate segment that configures that service. Do not view the I&RTS segment compliance criteria as roadblocks, but rather as guidelines. And never think in terms of a final delivery. ◆

References

1. Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification (I&RTS), Version 4.1, 3 Oct. 2000,<http://dod-ead.mont.disa.mil/cm/general.html>.
2. Defense Information Infrastructure (DII) Common Operating Environment (COE) Kernel Platform Compliance (KPC) Program Document for DII COE Kernel Version 4.2, p. 4, 31 May 2001, Version 1.1 (draft),<http://diicoe.disa.mil/coe/kpc/kpc_program_doc.doc>.

Notes

1. Information provided by the DII COE Engineering Office.
2. See <http://diicoe.disa.mil/coe/aog_twg /aog_twg_page.html> for a listing of TWGs and their home pages.
3. See <http://diicoe.disa.mil/coe/aog_twg/aog/members.htm> for a listing of AOG members.
4. See <http://diicoe.disa.mil/coe/kpc/KernelPlatformProgram.htm>.
5. The Compaq certificate is at <http://diicoe.disa.mil/coe/kpc/Compaq/19990831certNoSig.png> and the SGI certificate is at <http://diicoe.disa.mil/coe/kpc/sgi_cert.jpg>.
6. There is the dilemma that if any of the kernel source code was modified in order to get it to compile on the platform, then the specification has been modified and the platform is no longer running the government supplied source code.

About the Author

Gregory Frazier, Ph.D., has worked at Science Applications International Corporation (SAIC) since December of 1984. He was the integrator for Internet applications (the Web, newsgroups, Internet relay chat) for the Global Command and Control System and spent a year as the chief engineer as the DII Integration Contract for SAIC. For the last several years he has focused on research of enterprise computing for SAIC and is currently the software architect for the Joint Network Management System. He received a bachelor's degree in computer science and engineering from Massachusetts Institute of Technology and a doctorate in computer science from the University of California at Los Angeles.

8301 Greensboro
Mail Stop E-2-5
McLean, VA 22102
Phone: (703) 676-6459
Fax: (703) 676-7123
E-mail: gregory.frazier@saic.com

DII COE for Real Time: Becoming Reality

Lt. Col. Lucie M.J. Robillard
U.S. Air Force

Dr. H. Rebecca Callison and Marilynn B. Goo
The Boeing Company

John Maurer
MITRE Corporation

The Defense Information Infrastructure (DII) Common Operating Environment (COE) provides an environment in which common reusable infrastructure and applications across information systems help achieve goals for interoperability. The Department of Defense has been working for the past three years toward realization of a vision for extending these reuse and commonality initiatives to improve the effectiveness of systems performing real-time missions. This article describes the products, processes, tools, and techniques that have been developed to meet the needs of the integrator of DII COE-compliant real-time systems.

The following opening statement from a September 1999 CROSSTALK article began a presentation on the motivation for and objectives of an effort to extend DII COE to real-time systems:

> "The Defense Information Infrastructure (DII) Common Operating Environment (COE) originated with a simple observation about command and control systems: Certain functions (mapping, track management, communication interfaces, etc.) are so fundamental that they are required for virtually every command and control system. Yet these functions are built over and over again in incompatible ways even when the requirements are the same or vary only slightly between systems. If these common functions could be extracted, implemented as a set of extensible building blocks, and made readily available to system designers, development schedules could be accelerated and substantial savings could be achieved through software reuse. Moreover, interoperability would be significantly improved if common software were used across systems for common functions [1]."

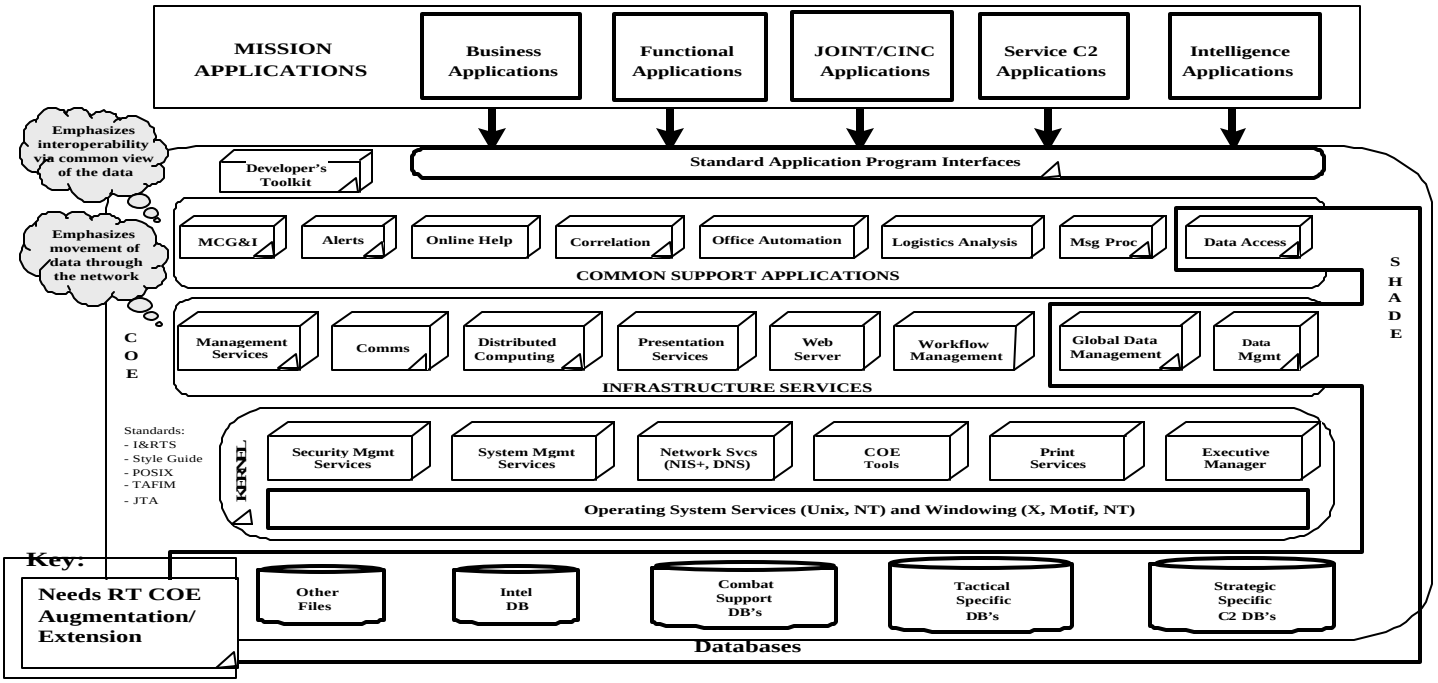
In this article we report on the status of those activities. Real-time processing is defined as a computation whose correctness depends on being logically correct and complete by a designated time. In a real-time system, the time that a process completes its computation and delivers its results is as important to correctness as, for example, the precision or accuracy of the answer. What is important is not *only* how fast the system responds but that it responds at the appropriate time. A protocol for synchronizing clocks across a communication network (distributed time service) is required to be accurate and timely, not just fast. The next generation of real-time systems will be so complex that more technical sophistication, not raw speed, will be the critical factor.

Extending DII COE for Real Time

In 1996 at the U.S. Air Force Electronic Systems Center, Hanscom AFB, Mass., all command and control programs were asked to develop a set of requirements for real-time extensions to existing DII COE capabilities. In the spring of 1997, representatives from the Air Force, Army, and Navy met to discuss the high correlation of real-time requirements across the services. In July 1997 these three services along with the Marine Corps jointly petitioned the Defense Information Systems Agency (DISA) to charter a DII COE Real-Time Technical Working Group (RT, TWG), with an aim of developing a set of common requirements and recommendations for potential products to provide real-time capabilities to the DII COE. DISA approved the services' request, and the Real-Time TWG began meeting in August 1997.

Initial studies conducted at Electronic

Figure 1: *Architectural Vision For DII COE Real-Time Extensions, August 1997*



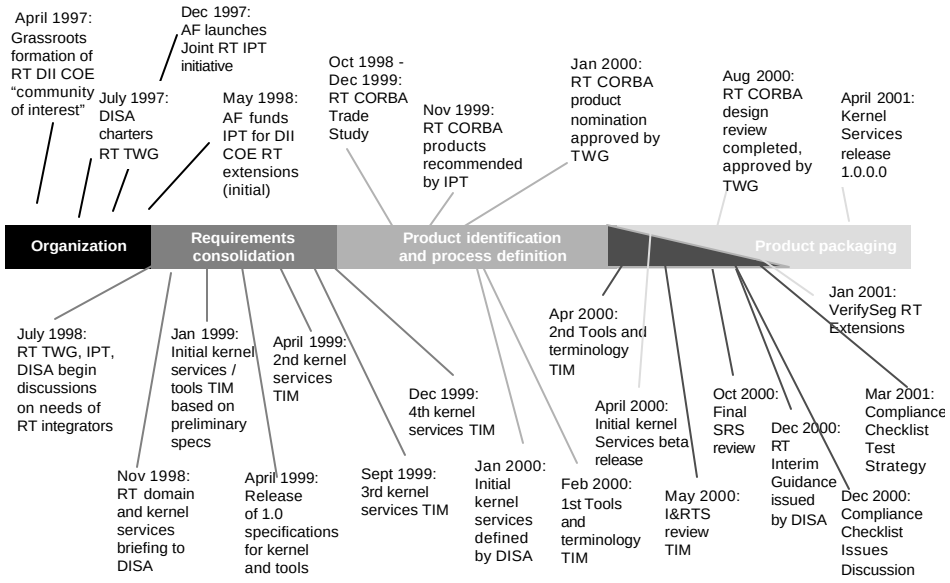


Figure 2: *Evolution Of DII COE Real-Time Extensions*

Systems Center highlighted numerous, relevant characteristics of real-time systems and suggested that a non-orchestrated approach to assembling real-time components would not be effective. In late 1997, the Air Force designated the Airborne Warning and Control System (AWACS) Program Office as Executive Agent for DII COE Real-Time Extensions. The DII COE Joint Real-Time Integrated Product Team (Joint RT IPT) embodies that executive authority. Because their missions are so closely related, the RT TWG and IPT have worked in continuous coordination, conducting joint meetings and sharing data. Both the TWG and IPT enjoy the active support and participation of Army, Air Force, Navy, and intelligence community representatives, all focused on the vision that is captured in Figure 1 (see page 19). The products, processes, tools, and techniques described in this article are the result of the activities of these two groups working in collaboration with DISA.

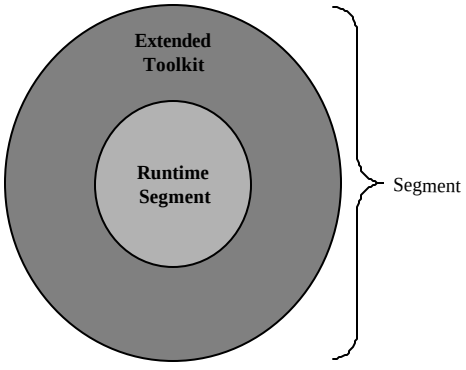
Real-Time Extensions

As depicted in the timeline of Figure 2, the anticipated release of a DII COE with real-time extensions represents the culmination of nearly four years of effort on the part of DISA, the RT TWG, and the Joint RT IPT.

Initially, the activity of the RT TWG focused on collecting requirements from the RT community and consolidating them in software requirements specifications. As an understanding of these requirements matured, the TWG engaged the DISA DII COE engineering office in translating the requirements into beta implementations of a DII COE kernel and segment development tools with real-time extensions. The RT TWG also worked with DISA to modify its processes to support inclusion of real-time components in the DII COE. In parallel, the RT IPT focused on finding and packaging components required by the real-time community to provide capability above the DII COE kernel.

The specifications developed by the RT TWG and submitted to DISA became the basis for the real-time kernel and real-time extensions to the segment development tools. In 2001, DISA's DII COE-wide kernel TWG assumed control of the RT TWG specifications for real-time kernel services and real-time extensions to the segment development tools. The RT TWG also specified requirements for tools to aid the real-time systems integrator. DISA's Toolkit TWG assumed responsibility for the real-time extensions to the integration tools specification.

Figure 3: *Extended Toolkit vs. Runtime Segment*



Evolution of DII COE Real-Time Extensions

Segmentation Concepts

Segmentation concepts encompass both the packaging constraints and the tools provided by DISA to ensure compatibility and peaceful coexistence among applications installed in a runtime (*mission*) environment. DISA's original segmentation concepts focused on executable binary applications and dynamically linkable libraries, i.e., the component formats that are used directly in the runtime environment of an operational system. This concept has been extended to support distribution of statistically linkable object libraries that may be combined with other components to produce tailored executable application images for the DII COE runtime environment.

Extended Toolkit Segments vs. Runtime Segments

Relating real-time requirements to the concept of segmentation resulted in a modified definition of segment, shown in Figure 3. *Segment* is defined by DISA in *Interim Guidance for DII COE Realtime Extensions* [2] as a "collection of one or more software and/or data units most conveniently managed as a unit of functionality." A runtime segment is a "segment that has been stripped of extraneous files and directories that are not required for a runtime target system."

A runtime segment corresponds roughly to the classic definition of a DII COE segment: software, data, and configuration information that will become part of and are used in the DII COE runtime environment. An extended toolkit is a "segment that contains documentation, shared libraries or those able to be linked, data, and other items required for use in an integration, development, and/or runtime environment."

The extended toolkit includes the runtime segment plus additional information needed to produce a runtime executable application. The extended toolkit enables DII COE software to be efficiently integrated into complex weapon systems in an integration environment prior to installing the software for operational use. This allows engineers to optimize DII COE compliant software for peak performance, a step that the existing DII COE did not previously support.

Extended toolkits and runtime segments are both *segments*. In general, a runtime segment is a proper subset of an extended toolkit segment. The classic DII COE runtime segment is delivered to a system integrator in the format accepted by the DII COE Installer tool. An extended toolkit is delivered in *tar* format and is loaded into the integrator's development

and/or integration environment using simple operating system utilities. It is important to note that the definitions of extended toolkit segment and runtime segment apply across the DII COE and are not unique to real time.

Figure 4 provides the directory structure of a generic extended toolkit segment. In this figure, a dotted line box contains the items that are also applicable to a runtime segment. The parts of the extended toolkit that are used in the integration environment are marked with diagonal lines. Similarly, the light colored boxes show directories that are used in the development environment and the darker boxes show directories that are used in the runtime environment.

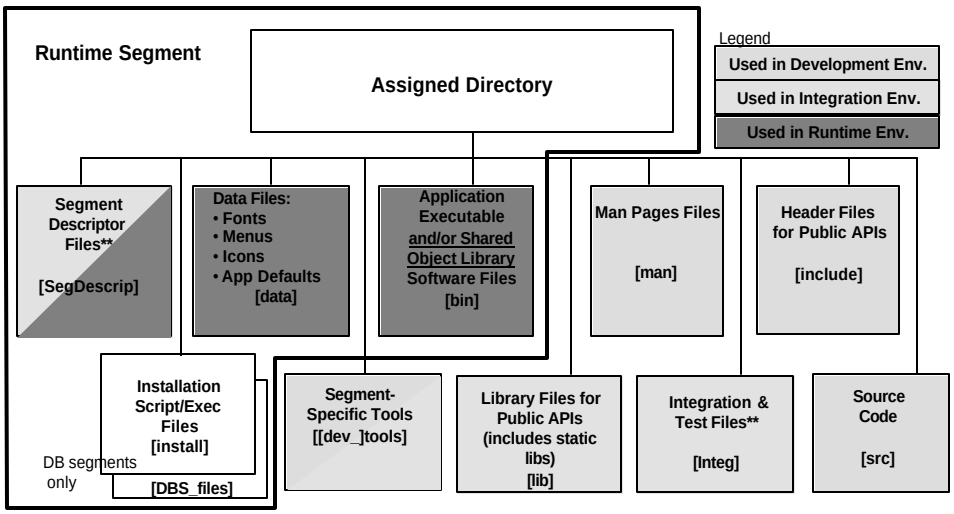


Figure 4: *Extended Toolkit Segment Directory Structure*

As an aid in the configuration of full systems from DII COE components, the dependencies of DII COE real-time segments on other DII COE segments, kernel services, and real-time operating system (RTOS) Portable Operating System Interface (POSIX)[1] units of functionality are documented in the segment descriptors provided by the segment supplier. A designer of a real-time system can then match system requirements to choices of DII COE applications, infrastructure, kernel services, and RTOS.

Additions Motivated by Real Time

A more in depth look at the segment contents uncovers additions, in Figure 5, that were motivated by the inclusion of real-time segments. Extensions to the SegInfo file in the SegDescrip directory describe additional hardware dependencies (e.g., shared memory and special hardware devices) and software dependencies (e.g., POSIX Units of Functionality) that may limit the execution environments in which segments can run. Beyond the addition of specific keywords to SegInfo, the attribute *<:restricted>*, when appended to *$CPU* or *$OS* identifiers, tells the integrator that the segment may place unusual restrictions on the operating environment. These restrictions must be fully documented in the IntgNotes file that is included in the Integ directory.

The IntgNotes file has been extended significantly to include items of key interest to integrators of real-time systems. Enhancements have two principal forms: freestanding additions and additions that explain or elaborate conditions noted in the extended SegInfo file. Examples of the former include IntgNotes entries under which real-time scheduling policies and restrictions, scheduling frequencies, jitter tolerance, CPU utilization, and other aspects of real-time behavior may be described. Examples of the additions which explain SegInfo entries include 1) entries under which <:restricted> notations applied to $CPU or $OS keywords are fully described and 2) entries in which the rationale for CPU, memory, and disk resource requirements are documented. The IntgNotes file remains a free-format text file into which the segment developer may insert any information deemed of potential interest to the system integrator using the segment.

As with the basic segmentation concepts, it is important to remember that none of the SegInfo or IntgNotes extensions are real-time specific. Some of this information is *required* in conjunction with real-time segments. For other segments, use of the extensions is optional.

Real-Time Products

Several products will be available for use in 2001 that provide real-time performance and are part of DII COE. Figure 6 (see page 22) shows how each of the products relates to Figure 1 (see page 19).

DII COE Real-Time Kernel and Platforms

The DII COE kernel provides the basic interfaces and functions to be used by standards-based infrastructure components and DII COE-compliant applications to achieve portability between systems. The DII COE configurable real-time kernel[2] extends basic DII COE concepts in two ways. First, the RT Kernel is hosted only on operating systems that provide real-time scheduling capabilities, reasonably predictable operating system performance, and the services required for timely execution of real-time tasks and processes. RT Kernel services are selectable, rather than mandatory.

Second, since real-time applications often need a very efficient operating system with small memory footprint for performance reasons, the design philosophy of the DII COE RT Kernel allows a system integrator to tailor the RTOS itself to meet system needs.[3] The RT Kernel is configurable and the integrator of a DII

Figure 5: *Process and Information Extensions*

DII COE Segmentation with Extended Toolkits

- **Guidance Documentation**
  - ✦ Augmented I & RTS
  - ✦ Interim Guidance for RT
- **Example Segments**
  - ✶ Add example RT segments
- **Segment Descriptor Files**
  - ✦ Enhanced SegInfo
  - ✦ Enhanced IntgNotes
- **Segmentation Tools**
  - ✦ Extended VerifySeg for new descriptions
  - ✶ Extended VerifySeg for RT attribute checks
- **Dependencies defined in SegInfo descriptors**
  - ✦ Segments/Toolkits
  - ✶ Real Time Operating System
  - • New HW Platforms
  - • Disk Space
  - • Memory
  - ✦ POSIX Units of Functionality
  - ✦ Custom Hardware Devices
  - ✦ Shared Memory
- **Installation Process**
  - ✦ Installation in Integration Environment (Not Runtime)
- **Segment Repository**
  - ✶ Port some DII COE Segments to RT COE

✶=extension used uniquely by RT segments
✦=extension motivated by real-time requirements

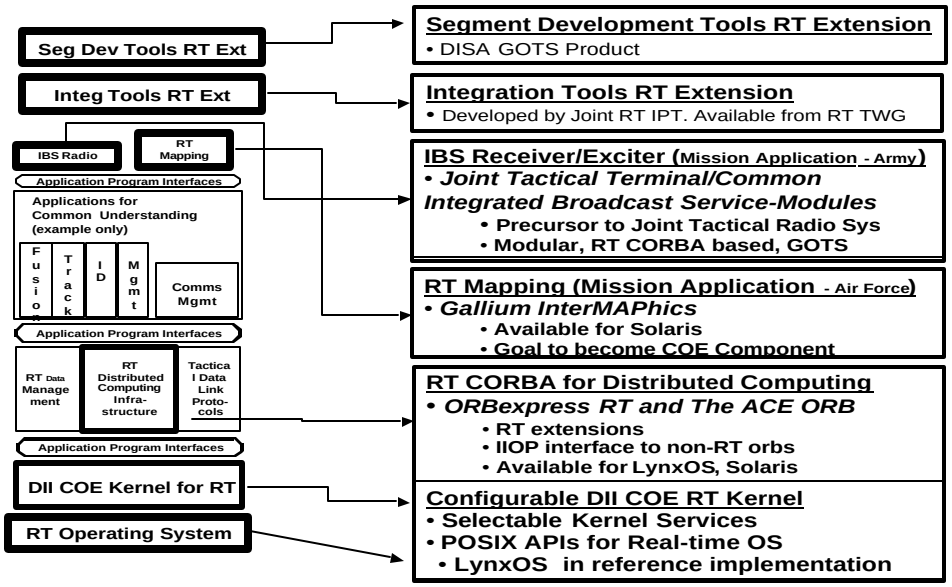


Figure 6: *Products Projected as DII COE Compliant Real-Time Segments*

COE-compliant system tailors the RT Kernel by selecting only those services required for the specific computing configurations of the target system. POSIX Application Program Interface (APIs) for operating system services, including APIs for threads and real-time extensions specified in [3], form part of the RT Kernel API. Each RTOS being considered for use in the DII COE will be assessed for its ability to provide key functional units associated with real-time profiles in the POSIX 1003.13 standard [4]. LynxOS[4] Version 3.1.0a running on PowerPC was selected as the reference (i.e. first) implementation. Sun Solaris[5] Version 8 is the second real-time capable operating system on which the RT Kernel is hosted.

As noted earlier, the RT Kernel has two parts: 1) a RTOS with POSIX application program interfaces and 2) selectable DII COE kernel services for real time. The RT Kernel services are provided by DISA. DII COE for real time includes X, Motif, and Domain Name Server, all of which are commercial off-the-shelf products, plus government off-the-shelf services for system startup and shutdown, setting system time, and starting and stopping DII COE processes. Requirements for the RT Kernel services are documented in [5].

CORBA Infrastructure for Real Time

Common Object Request Broker Architecture (CORBA) is an international standard [6] for distributed computing that is governed by the Object Management Group. The CORBA standard provides for flexible interconnection of objects in a client-server model for distributed computing. Four key objectives of CORBA are support for location independence, operating system independence, hardware independence, and language independence in the design of software components.

Additions to the CORBA standard to enable real-time computing with end-to-end predictability are documented in the CORBA specification revision 2.4 [6], which was formalized by the Object Management Group in October 2000. These additions allow for the association of real-time priorities with tasks and requests, the passing of priority information between communicating components, and the capability to express and monitor timing constraints for requests. These additions also define a scheduling service that provides a consistent real-time scheduling model across a CORBA-based system.

In 1999, a real-time CORBA trade study was performed by the Joint RT IPT to assess products being considered for use by the real-time community. The goal of the study was to determine whether or not any or all of these products would be suitable for use in the real-time extensions of the DII COE. The study included five assessments. The technical assessments addressed three areas: standards compliance, basic performance, and interoperability with other object request brokers (ORBs). The other assessments included a user survey of product usability and a cursory examination of the business viability of each vendor and product. At the time of the assessments, the real-time additions to the OMG CORBA standard were still in development and none of the vendors had implemented them. The study can be obtained from <www.hanscom.af.mil/foia/misclist.asp?contractid=3&description=Other+Documents>.

Based on recommendations from the real-time CORBA trade study, the RT TWG nominated two real-time ORBs for inclusion in the DII COE infrastructure.[6] The real-time ORBs are ORB*express* RT[7], a product of Objective Interface Systems, Herndon, Va., and The ACE ORB (TAO)[7], an open source product of Washington University, St. Louis, Mo., that is commercially supported by Object Computing, Inc., also in St. Louis. ORB*express* RT supports both Ada and C++ and includes extensions for real-time performance. TAO supports C++ and also includes extensions for real-time performance. Both are cognizant of real-time request priorities and provide the capability to associate deadlines with requests. Vendor packaging of these products as extended toolkit segments began in 2000 and was completed in mid-2001.

Real-Time Mission Applications: A real-time mapping product is available as a DII COE-compliant AF mission application during 2001: InterMAPhics[8], a product of Gallium Software, Inc., Nepean, Ontario, Canada. This product may someday be in the DII COE Common Support Applications layer. The decision to proceed in this direction depends on the capabilities of the product selected by the National Imagery and Mapping Agency under the Commercial Joint Mapping Toolkit procurement. In the meantime, the real-time community can use this high performance product as a real time alternative to the Joint Mapping Visualization part of Joint Mapping Toolkit.

The Army is developing a software programmable exciter/receiver called the Joint Tactical Terminal/Common Integrated Broadcast Service-Modules (JTT/CIBS-M) under contract with Raytheon, St. Petersburg, Fla. This software is being packaged as a DII COE-compliant Army mission application during 2001. JTT/CIB-M supports a variety of intelligence broadcast protocols such as TDDS, TADIX-B, TIBS and TRIXS.

DII COE Real-Time Tools

Based on the RT TWG specification for real-time extensions to segment development tools, DISA has modified its VerifySeg tool to check the additions to the SegInfo file segment descriptor information described earlier in Figure 5 (see page 21). This version of VerifySeg is used by developers of runtime segments as well as by developers of extended toolkit segments. The output from VerifySeg becomes part of the segment. VerifySeg and the requirements for its use are

described in DISA's *Integration and Runtime Specification (I&RTS)* [7].

In the fall off 2000, the Joint RT IPT began developing the real-time integration tools to prototype the integration process associated with developing a DII COE compliant real-time system. These tools are intended for use by software system integrators in the integration environment. Since the COE installer tool cannot be used to install runtime segments in many real-time runtime (target) environments, some automated assistance is needed for ensuring a proper configuration of DII COE segments. The real-time integration tools provide this assistance by analyzing the intended segment configuration for inter-segment dependencies and conflicts. For embedded systems, they also assist the integrator in configuring (scaling down) the operating system (OS) to include only those OS functions needed to support the target application software load.

The integrator supplies a list of the capabilities to be configured on a target system by selecting from a list of available segments. The primary output of the real-time integration tools is a real-time configuration (RTConfig) file that lists all segments, including kernel services that must be loaded on the target platform in order for the selected segments to run. Using information contained in segment SegDescrip directories, the real-time integration tools expand the list of selected functions based on the dependencies of runtime and extended toolkit segments on other segments. For example, if segment A depends on segment B, and segment B depends on segment K, then the RTConfig file will include segments A, B, and K. The real-time integration tools also produce a list of POSIX capabilities (e.g., POSIX units of functionality) required by these segments in the OS configuration. If any of the segments have conflicts noted in their SegInfo files, this information is included in the RTConfig file.

The real-time integration tools were completed in June 2001. Acceptance of these tools by DISA depends on the customer demand for the tool and the value added seen by DISA. In the meantime, the RT TWG makes the tools available to interested users upon request via the RT TWG Web page <www.dii-af.hanscom.af.mil /infrastructure/COE/TWG/COE/TWG/rtcoe/NewTWG/index.htm>.

DISA Process

Real-Time Interim Guidance

The new capabilities that are available with the DII COE real-time extensions have required that the rules governing the development of DII COE segments be modified. These new capabilities include extensions to exploit the features of a real-time platform, the configurable kernel, and development of extended toolkit segments intended for delivery to an integration environment. The existing rules are documented in DISA's *Integration and Runtime Specification (I&RTS)* [7]. The modifications to these rules have been published in the *Interim Guidance for Defense Information Infrastructure Common Operating Environment (COE) Realtime Extension* [2], which will be refined through initial practical experience and incorporated into Version 5.0 of the I&RTS.

The interim guidance document provides detailed information that a segment supplier needs to develop segments intended to run on the RT Kernel. In addition, the interim guidance document provides a preview of how the rules may be applied in the next major release, DII COE Version 5.0. The interim guidance document provides a discussion of definitions and concepts, as well as an updated version of the compliance checklist criteria (aka, Appendix B) as it applies to the real-time platform. It is available in the technical baseline section of the RT TWG Web site <www.dii-af.hanscom.af.mil/infrastructure/COE/TWG/COE/TWG/rtcoe/NewTWG/Baseline/DIICOERTEInterimGuidance.PDF>.

Toolkit Compliance Evaluation

Compliance evaluation for an extended toolkit requires a different perspective than for a classic DII COE runtime segment. The latter is installed directly in the runtime (target) environment, whereas the former is not – it is loaded first into an integration environment. In both cases (runtime segments and extended toolkit segments), compliance evaluation is intended to ensure correct *runtime* behavior. For runtime segments, the evaluation can be performed in the target (runtime) environment. However, in the case of extended toolkits, the compliance evaluation must, in general, be performed in a development or integration environment with an eye toward how a *runtime* segment *built from the extended toolkit* will behave in the *runtime* environment.

When extended toolkits contain static libraries only, there is no clearly identifiable *runtime segment* subset of the toolkit that will *ever* be installed directly into the runtime environment. Rather the integrator will *always* link the libraries of these toolkits with other applications and toolkits to produce the target system executables. However, in order to ensure that the integrator's customized executables satisfy the constraints of the DII COE environment, the behavior of the delivered libraries, as contributors to that behavior, must be scrutinized. For example, a number of compliance criteria address constraints on the creation of files outside the DII COE directory structure. If an application links to a library that creates files in a non-compliant location, the application executable can never be compliant. Since each compliance criterion that affects the application executable must be flowed down to the library in the extended toolkit as well, the problem becomes one of defining the details of how to evaluate compliance for toolkits.

After the release of [2], the RT TWG was asked to determine for each item in the interim guidance whether or not that Appendix B compliance checklist item applies to an extended toolkit segment. The RT TWG was also asked to determine the test strategy for each applicable item. This action was completed in early 2001. The resulting document, which is available at <www.dii-af.hanscom.af.mil/infrastructure/COE/TWG/COE/TWG/rtcoe/NewTWG/baseline.htm>, is the RT TWG's recommendation for ensuring correct runtime behavior while performing compliance evaluation of extended toolkits in an integration environment.

A Turnkey Segmentation Process

Segmenting software for the DII COE is a recurring task where attention to detail can prevent rework. The RT IPT has been working with various services, agencies, DISA, and segment suppliers to define the steps necessary to enable real-time products to become DII COE compliant. Lessons learned as a result of this effort are being recorded in the form of tutorials and templates. Table 1 is a partial list of prod-

Tutorials	Templates
Segmentation Overview	DISA Design Review Questions
Segmentation	Segmentation Plan
Prefix & Segment Registration	IntgNotes

Table 1: *Turnkey Segmentation Products*

ucts available from the RT TWG for use by those who need to prepare extended toolkit segments for the DII COE.

DII COE Real-Time's Future

In 2001, several changes have occurred regarding the future of this effort. DISA has changed the name of the RT TWG to the Real-Time Advisory Group (RTAG). There will be slight modifications to the charter for this group. In addition, the Joint RT IPT has completed its tasks and transitioned its remaining responsibilities to the RTAG (as of 30 June 2001). So the RTAG will be the main point of contact for the real-time community regarding DII COE capabilities and requirements.

The RTAG remains under the umbrella responsibility of the DISA DII COE Chief Engineer Office. The chairman of the RTAG remains on the DII-Air Force Office staff at Hanscom AFB, Mass., and continues to serve all services and agencies.

Based on the work of the Joint RT IPT and RTAG in the last four years, it is envisioned that the future products needed by the real-time community in the DII COE include real-time data access, Link 16, Joint Variable Message Format (JVMF) Parser, Data Correlator, and Alerts. It is the responsibility of each system program office and their sponsoring service or agency to work with the RTAG to sponsor appropriate government or commercial off-the-shelf products into the DII COE. This responsibility will often include performing the work needed to create a segmented product according to DISA's I&RTS [7]. The turnkey segmentation package created by the Joint RT IPT is a valuable resource in accomplishing this task.

Conclusion

In four years, a small contracted engineering effort supplemented by a strong team of engineers working on donated funding have accomplished key tasks that make it possible for real-time weapon systems to pursue increased command and control interoperability via DII COE compliance. The basic vision has been accomplished.

The Joint RT IPT provided solid, reliable technical leadership and support for a fast-paced, dynamic program fraught with technical challenges. The real-time community can now directly utilize the lessons learned from this effort to bring more software products into the DII COE. The foundation has been set for more work to be done in support of the real-time community by working with the RTAG. As the real-time effort is normalized into the mainstream of service acquisition agencies, the future enhancements of DII COE for real time lie in the hands the real-time weapon system builders and their customers, and the services and agencies with real-time command and control missions.◆

References

1. Robillard, Lt. Col. Lucie M.J., Dr. H. Rebecca Callison and John Maurer. "Extending the DII COE for Real-Time," CROSSTALK, Sept. 1999.
2. Interim Guidance for Defense Information Infrastructure Common Operating Environment (COE) Realtime Extension, Version 1.0, Defense Information Systems Agency, Dec. 2000.
3. "Portable Operating System Interface (POSIX) Part 1 – System Application Program Interface (API) [C Language]," *Information Technology*, ISO/IEC 9945-1:1996 (E) ANSI/IEEE Std. 1003.1.
4. Portable Operating Systems Interface (POSIX) 1003.13-1998, IEEE Standard for Information Technology Standardized Application Environment Profile (AEP)–POSIX Realtime Application Support.
5. Software Requirements Specification for Kernel Services for the Real-Time Defense Information Infrastructure Common Operating Environment (RT DII COE), Revision 1.5, DII COE RT TWG, 30 Jan. 2001, <www.dii-af.hanscom.af.mil/infrastructure/COE/TWG/COE/TWG/rtcoe/NewTWG/baseline.htm>.
6. The Common Object Request Broker: Architecture and Specification, Revision 2.4, Object Management Group, Inc., Oct. 2000, <http://cgi.omg.org/cgiin/doc?formal/00-10-01>.
7. Defense Information Infrastructure (DII) Common Operating Environment (COE) Integration and Runtime Specification, Version 4.1, DISA, DISA Joint Interoperability and Engineering Organization, Reston, Va., Aug. 2000.

Notes

1. POSIX is a registered trademark of The Institute of Electrical and Electronic Engineers, Inc. (IEEE).
2. In the rest of this paper, we will use the term "RT Kernel" as an abbreviation for "DII COE Configurable RT Kernel."
3. COTS tools provided by the OS vendor are used to configure the OS, not DII COE unique software. The degree to which a specific RTOS can be configured depends on the flexibility provided by the RTOS vendor.
4. LynxOS is a trademark of LynuxWorks, Inc. <http://www.lynuxworks.com>.
5. Sun and Solaris are trademarks of Sun Microsystems, Inc. <http://www.sun.com>.
6. ORBexpress RT is a trademark of Objective Interface Systems <http://www.ois.com>.
7. The ACE ORB is a trademark of Douglas Schmidt, Washington University, and the University of California, Irvine <http://www.cs.wustl.edu/~schmidt/TAO.html>.
8. InterMAPhics is a trademark of Gallium Software, Inc. <http://www.gallium.com>.

About the Authors

Lt. Col. Lucie M.J. Robillard is the Air Force executive agent for Real - Time DII COE at Hanscom AFB, Mass. She is the chair for the DII COE Joint Real-Time Integrated Product Team that has the charter to make real-time extension to DII COE a reality for all services. She is a Level III certified acquisition professional with Joint assignment experience. A majority of her assignments have dealt with software acquisition and engineering. She has a bachelor's degree in electrical engineering from the University of Vermont and a master's of science degree in systems management from University of Southern California. As of July 1, 2001, she has moved on to another assignment.

ESC/AWPD
3 Eglin Street
Hanscom AFB, MA 01730
Phone: (781) 377-2679
E-mail: lucie.robillard@hanscom.af.mil

H. Rebecca Callison, Ph.D., is the software architect for the U.S. Airborne Warning and Control System (AWACS) Block 40/45 Program. Previously, she was the lead for the Boeing team supporting Lt. Col. Robillard and the DII COE Joint Real-Time Integrated Product Team. Dr. Callison has 25 years of experience in the design and implementation of real-time systems, principally in the area of defense systems. She has a bachelor's of science degree from the University of South Carolina, a master's of science engineering from the University of Pennsylvania, and a doctorate from the University of Washington. Her research interests focus on concurrency control in real-time systems.

The Boeing Company
P. O. Box 3999, MS 81-75
Seattle, WA 98124
Phone: (253) 657-3952
Fax: (253) 657-4269
E-mail: rebecca.callison@boeing.com

Marilynn B. Goo is the program manager for the Boeing team supporting Lt. Col. Robillard and the DII COE Joint Real-Time Integrated Product Team. Goo has 26 years of experience in design and implementation of complex systems, principally in the area of defense systems. She has a bachelor's of science in mathematics from the University of Washington and an master's of business administration from Stanford University.

The Boeing Company
P.O. Box 3999, MS 82-84
Seattle, WA 98124
Phone: (253) 773-9867
Fax: (253) 657-2892
E-mail: marilynn.b.goo@boeing.com

John Maurer leads MITRE's Real-Time and Performance Engineering Section. Maurer is also the chairperson of the DII COE Real-Time Advisory Group. He has a bachelor's of science in mechanical engineering from Massachusetts Institute of Technology and 24 years experience implementing software-intensive Department of Defense systems. His work experience includes real-time system development for airborne surveillance systems and Army vehicle systems.

MITRE Corporation
202 Burlington Road
Bedford, MA 01730
Phone: (781) 271-2985
Fax: (781) 271-4686
E-mail: johnm@mitre.org

Coming Events

November 4-7
Amplifying Your Effectiveness Conference 2001
Phoenix, AZ
www.ayeconference.com

November 4-8
ASIS 2001 Annual Conference
Washington, D.C.
www.asis.org

November 6-8
TechNet Asia-Pacific 2001
Honolulu, HI
www.afcea.org

November 12-16
5th International Software and Internet Quality Week - Europe 2001
Brussels, Belgium
www.qualityweek.com

November 13-15
1st Annual CMMI Technology Conference and User Group
Denver, CO
djenks@ndia.org

January 27-31, 2002
2002 Western Multiconference
San Antonio, TX
www.scs.org

February 4-6, 2002
International Conference on COTS-Based Software Systems (ICCBSS)
Lake Buena Vista, FL
www.iccbss.org

February 25-27, 2002
15th Conference on Software Engineering Education and Training (CSEE & T)
Covington, KY
www.spsu.edu/oce/cseet2001

March 19-21, 2002
Federal Office Systems Exposition 2002
Washington D.C.
www.fose.com

April 28 - May 2, 2002
Software Technology Conference 2002 "Forging the Future of Defense Through Technology"
Salt Lake City, UT
www.stc-online.org

Mission-Based Incremental Development of C2 Systems for More Efficient Business Support

Ingmar Ögren
Tofs Inc.

It is not possible to ever know enough about requirements before developing a Command and Control (C2) system. Also, a C2 system is never identical to its automated (computerized) parts. This article will introduce you to dominant battlefield awareness (DBA), how it is used to define a mission "Build DBA," and how abilities are connected to mission completion. Mission supports such as operator role, software, and hardware are used to model a complete C2 system, including four system levels: reality, computer information, presentation, and mental. The model is a useful basis for various simulations required during system development and evolution, as well as for acquisition of system components. Lastly, the possibility of extending the model to cover the domain of C2 systems is presented along with how that extension can become a basis for more rational production of C2 systems within the domain.

The development and evolution of Command and Control (C2) systems have suffered from problems, some great enough to cause project failures. Two erroneous assumptions have contributed to this situation:

- That contracts can be based on the premise that a C2 system can be sufficiently known prior to development to produce complete and high quality requirements specification.
- That a C2 system is identical to the automated (computerized) support for such a system.

What is wrong with these assumptions?

We know from experience that new knowledge will be gained during C2 system development that will cause changes to the original specification. Areas affected include system usage, system environment, system features usefulness, new mission completion possibilities, etc.

The goal of every C2 system is to complete one or more missions through a combination of human operator tasks and automated support. Consequently it is important to understand how the system completes its missions and how to manage human and automated systems in the same context.

With this in mind, two better assumptions would be the following:

- Prior to developing a C2 system, identify its missions and expect the detailed requirements to surface during development.
- To complete their missions, a C2 system requires operators and cooperating software and hardware.

To fully understand these assumptions, we need an example.

Dominant Battlefield Awareness

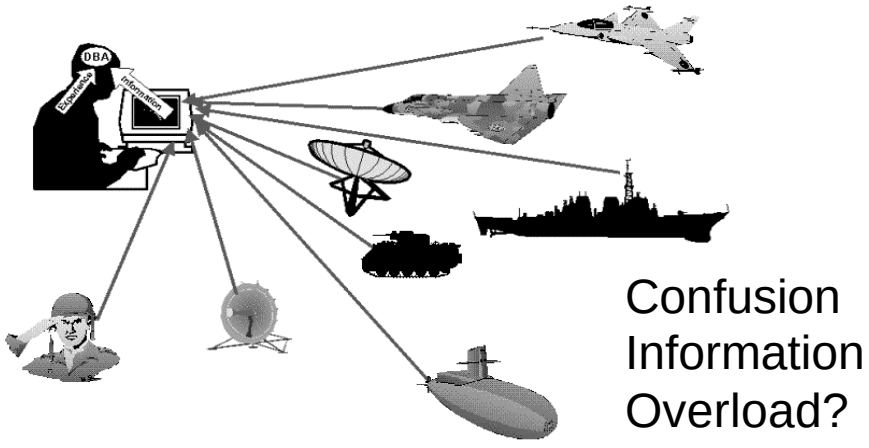
Dominant battlefield awareness (DBA) means that a commander in a C2 system builds an awareness of the battlefield situation by using many information sources such as agents, sensors, reports from other C2 centers, etc. Since DBA is an important part of the ongoing revolution in military affairs, the mission *Build DBA* is selected as an example to study system management.

Figure 1 shows that if all the information required to establish DBA in a real battle situation is linked directly to the commander two problems may surface:

- Confusion as a result of depending on data of different ages, from different origins, of differing quality, etc.
- Information overload through presentation of more data than is humanly possible to overview and understand.

The conclusion is that there is more to building DBA than to present all available information to the commander.

Figure 1: *Dominant Battlefield Awareness Requires Large Amounts of Information*



Connectabilities to the Mission

Completion of the mission Build DBA requires a set of abilities. An approach defining the necessary abilities in connection with the mission object Build DBA is shown in Figure 2. The abilities listed are:

- Analyze the situation, including interpreting the data in the context of known behavioral patterns, for units in the battlefield.
- Collect data from hostile units, including sensor, agent, report, and other data about hostile units.
- Control data processing in building DBA to support the commander's objectives.
- Control presentation(s) of commander and others involved to support building necessary awareness.
- Fuse data to avoid double presentation and enhance information validity.
- Maintain communication ensuring that all other friendly units' information remains available.
- Present information database for the commander and others in a C2 unit.

- Survey individual unit so that information is presented in the battlefield context keeping the commander aware of the situation.

What is important is that the abilities required for mission Build DBA can be seen as actions offered by a mission object, and consequently drawn in an object graph as shown in Figure 1.

Support Abilities with System Components

Now that we have defined a set of abilities, we need to create a system of components that actually has these abilities. In Figure 3, the mission object Build DBA is supported by a set of objects needed to build the abilities listed above. The support objects categories are operator (role), software, and hardware with examples:

- The "Commander" is an operator role object.
- The "Person-Machine Interface" and the "information base manager" are examples of software objects.
- "Data processing resources" is an example of a hardware object.

The diagram in Figure 3 is called a tree graph. It shows the need-lines between the objects in a system model. The version of the tree graph shown is drawn in the Tofs software tool, which also shows completion status for the different objects as little clocks.

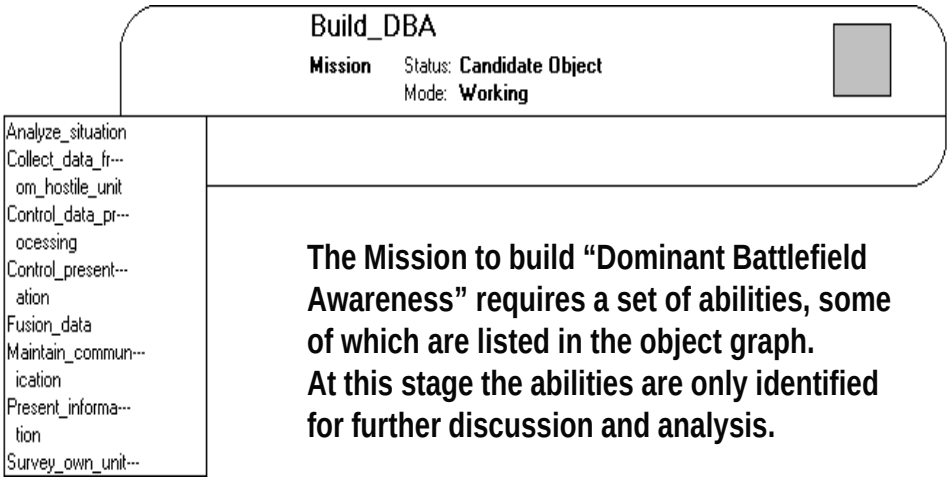
The Four Knowledge Levels

After building a system outline, you must consider what really constitutes DBA. The basic prerequisite is that the commander's mental awareness must comply with the actual battlefield situation.

Figure 4 (see page 28) shows the four knowledge levels necessary to understanding Build DBA, and which can be seen in a structured system model as shown above. The levels from the bottom are:

- The reality level representing battlefield reality or God's view.
- The computer information level representing all the data about the battlefield situation available in the C2 unit's computer system.
- The presentation level representing the information presented to the commander after data processing and selection.
- The mental level representing the commander's awareness after the presented information is combined with his personal experience and intuition.

It is obvious that the mental level must comply with the reality level to achieve DBA. The prerequisites for the required



The Mission to build "Dominant Battlefield Awareness" requires a set of abilities, some of which are listed in the object graph. At this stage the abilities are only identified for further discussion and analysis.

Figure 2: *Military Systems Operational or Being Developed Using the DII COE*

compliance can be studied in a system model built as a dependency structure as shown above.

Incremental Development with Simulation–Based Acquisition

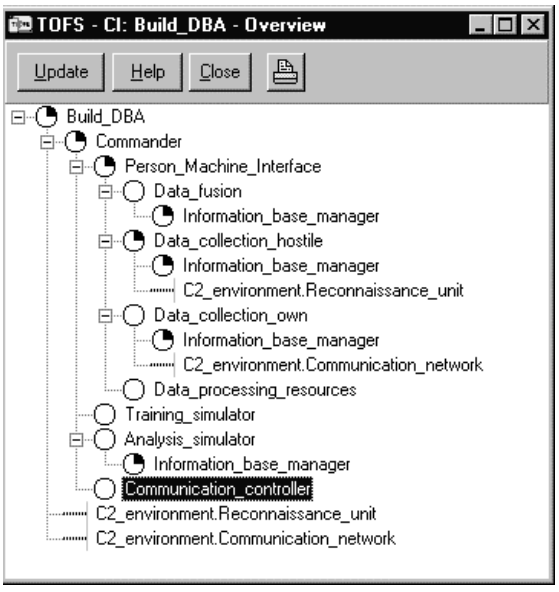
As stated above, it is not really possible to build a qualified C2 system from frozen requirements specification. Knowledge will inevitably grow during development, and some of this new knowledge will influence the requirements. Experience supports this insight since all non-trivial real C2 systems needed updates during development and repeated updates after the first delivery.

We conclude the need for an orderly way to manage changing requirements, to build and save new knowledge, and to change the system incrementally, especially as requirements insight grows and as the changing environmental situation introduces new requirements.

One way to do this is to use a model, as outlined above, as a system backbone. Use simulations to verify the model and investigate requirements, then let the model evolve incrementally. It will then be the system reference and a basis for both simulations and system component acquisitions.

Figure 5 (see page 28) shows a model used as a system reference, and how the model evolves through system increments. It illustrates how the project starts with an idea, how documentation can be connected to the model, and how the model is used as a basis for both simulations and system products (acquired components). Note that not only the system concerned needs to be simulated, but also its environment.

Figure 3: *A Model Outlined From the Mission Object "Build Dominant Battlefield Awareness" to Show the Components Needed*



The model is built from the assumption that systems are best modeled using the relationship "depends on".
The components are of categories:
- **Mission**
- **Operator (role)**
- **Software**
- **Hardware**

The "clocks" indicate completeness status for each component (object).

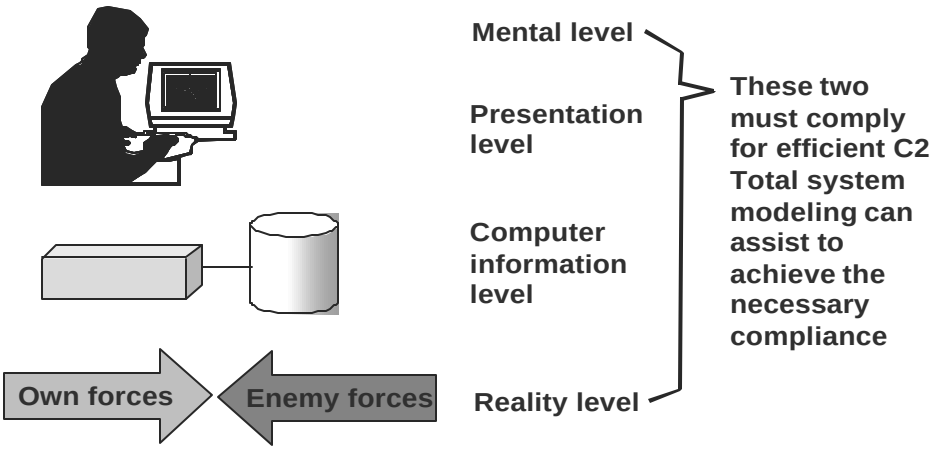


Figure 4: *The Four Knowledge Levels and the Necessary Compliance*

Formalize the Model

For a model to be a firm basis for multiple simulations and various system component acquisitions, it is essential that different system parts comply with each other and that the simulations comply with the system built. For example if the C2 system is of a non-trivial size, it must be possible to use a computer to check the model's consistency. The conclusion is that the model must be expressed in a formal syntax. Furthermore, this formal syntax must be readily understood by all those involved – developers, end-users, quality people, etc.

One way to achieve the required formality is to express the model in formal English using a limited language that includes:

- Reserved words to express control constructs.
- Variables of defined types.
- Comments that are used as explanations and to describe parts of the model that are not yet formalized.

The interaction between a commander and his Person-Machine Interface (PMI) to manage surveillance resources can be modeled as two concurrent processes. The commander's manual process interacts with the concurrent PMI software process through sending and receiving messages. An example of such a message is a presentation of a screen image that means the PMI has sent a message to the commander.

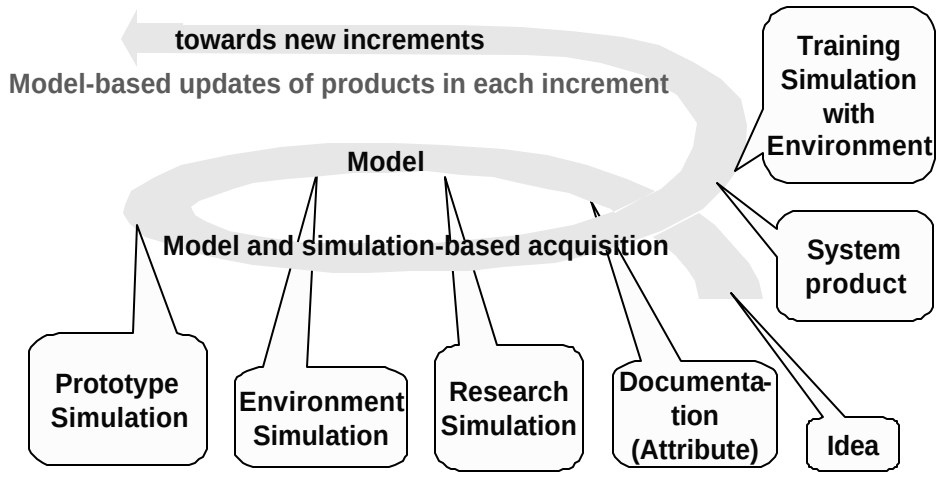
Generalize the Model to the Domain Level

We have discussed how to use a model as a backbone for incremental development of C2 systems, and how it obtains compliance between system simulations and system implementations. Since the history of C2 systems includes some reinventing of the wheel, you may wonder: Can the modeling technique be used to avoid such reinvention? Models can be used in combination with so-called domain engineering to minimize reinvention. A possible principle is:

- A similar set of C2 systems are analyzed and used as a basis for a generic architectural model for the C2 application domain. The architectural model can then be formalized as described above.
- The analysis result is further used to identify reusable assets from the existing C2 systems. These assets are connected to the relevant objects in the architectural model to prepare for reuse. Assets may then be, for example, software modules, requirements, manuals, specifications, hardware products, etc.
- For each new or modified C2 system, you begin by combining the original requirements specification with the domain architecture to create a tailored version of the domain architecture to satisfy the specification (possibly modified after the analysis work).
- As far as possible, the new system is implemented using assets connected to the domain architecture. Some components will normally have to be developed anew. These new components may then be turned into reusable assets.

The result is an orderly development process that, provided a number of C2 systems with some similarities are to be produced, will decrease cost and development time and increase quality.

Figure 5: *Using a Model for Iterative Development and Acquisition of C2 Systems*



Possible Objections

The principles described above may seem foreign and objectionable. However, if you take the Swedish Air Defense C2 system Stril 60, for example, it evolved from the early sixties into the nineties. This evolution was possible through cooperation within a group of technical and tactical experts with a good understanding of the system's missions, abilities, and structure. This suggests simply that the well-proven informal work, based on the informal understanding within a small expert group, can be supported by a formalized and commonly accepted system model.

There may still be objections to the model-based principle, but these are not too well founded:

- *We use specifications, not models for acquisitions.* Fine, but when you need more information than can be managed in a textual specification, why not supplement the written specification with a computer-based model?
- *You cannot model intuition, and our C2 systems depend on the experienced commander's intuition.* This may be true, but it can still be worthwhile to model all the routine details of the commander's interaction with systems for com-

puting and communicating in order to simplify such interaction. The result may be that the commander pays less attention to the support systems, consequently getting more time for his essential creative tasks.

- *You must separate tactical application development from technical acquisition*. Yes, this is traditionally what is done and may be one reason for experiencing problems with C2 system development. A working C2 system requires smooth cooperation between manual and automated parts. This smooth cooperation is best achieved through modeling the complete system as a single structure.
- *Computer-based modeling is just an expensive way to replace documentation.* The computer-based model will be a good basis for documentation and help make sure the documentation produced is really compliant with the system built or simulated. Using the model as a basis for documentation may consequently decrease documentation costs.

```
begin
      send Get_map(current_area)                          Commander's behavior
      receive Present_map(current_area)
      receive Present_surveillance_resources(available_resources)
      for resource in 1..total_resource  loop
           send Select_resource(resource_id)
           receive Present_adjustment(possible_adjustment)
           send Adjust(current_adjustment)
      end loop
end

begin
     while Commander_messages_received  loop
          receive Get_map(area)                          PMI software behavior
          Support_software.Get_area_map
          send Present_map(area)
          send Present_surveillance_resources(area)
          {The map information collected from the support software includes
          information about surveillance resources availabe in the area selected}
          receive Select_resource(Resource_id)
          send Present_adjustment(Possible_adjustment)
          {For each surveillance resource possible adjustment is received together
          with the map information}
          receive Adjust(Current_adjustment)
          Data_collection_own.Adjust_data_collection
          {The surveilance resource concerned is adjusted as required}
     end loop
end
```

Figure 6: *Formalization of the Interaction Between a Commander and His Person-Machine Interface to Manage Surveillance Resources*

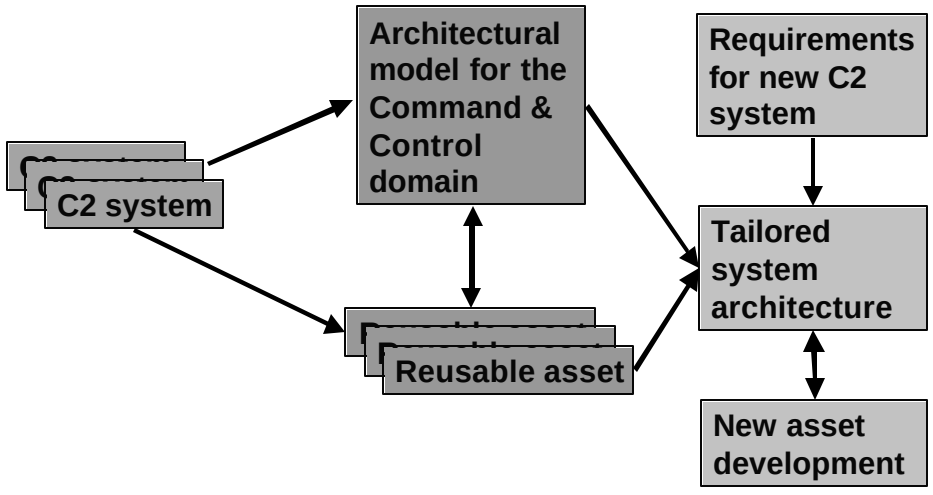
Conclusions

Obviously C2 system evolution should start from a system's missions and abilities. It has been shown that model-based incremental technique for C2 development work has some advantages:

- Computer-based models can be used as a common base for simulations, acquisitions, recruitment, and training. While each system is too large for one person to overview, the model will help ensure that everyone involved knows his or her work is interconnected with the rest of the project through the common model.
- Models can help manage and structure large amounts of information that are traditionally stored as paper documents. This reduces cost and ensures that available documentation complies with the current system version or simulator. Using a model with good configuration management makes managing that information easier; relevant information for the current system version or simulator is extractable from the model.
- Models can be used to create a back bone in incremental development of C2 systems. Since the model lies behind each system version and simu lator built to support development and training, it is a backbone for the devel opment and reengineering work. This will help ensure that different system versions and simulators really comply with each other.

In summary, model-based development and evolution means that well-proven principles are formalized and extended to cover larger systems with less dependence on expert groups.◆

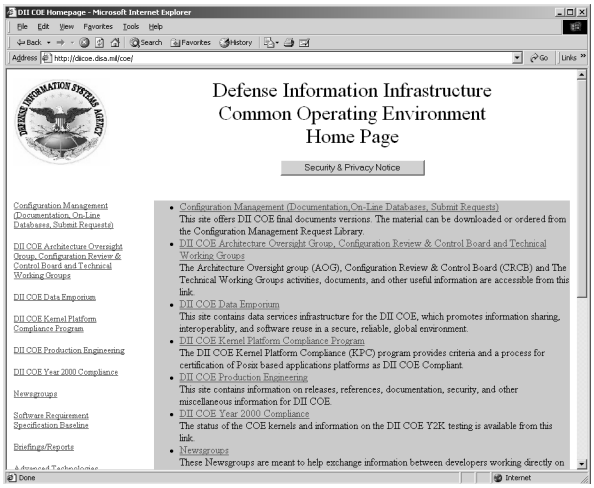
Figure 7: *Principles for Creating and Using a Domain Model*



About the Author

Ingmar Ögren has a master's of science in electronics from the Royal University of Technology in Stockholm. He has worked with the Swedish Defense Material Administration and various consulting companies in systems engineering tasks associated with communications, aircraft, and command and control. He is currently partner and chairman of the board for Tofs Inc. and Romet, a systems engineering consulting company mainly utilizing method O4S. He also teaches systems and software engineering. Ögren is a member of Modeling and Simulation in Sweden and International Council Of Systems Engineering.

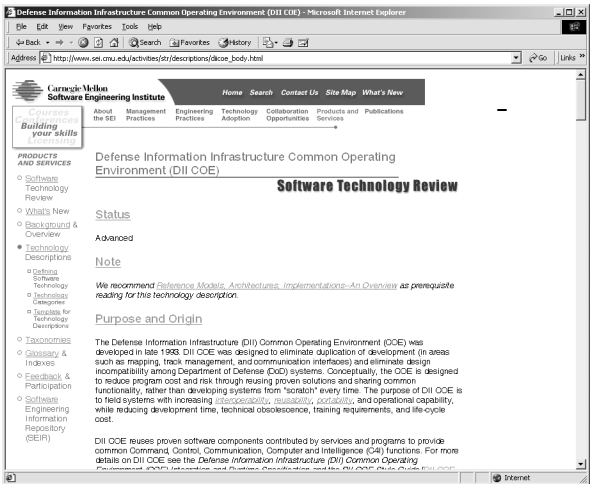
Tofs AB
Fridhem 2
S-76040 Veddoe, Sweden
Phone: (+46) 176-54580,
Fax: (+46) 176-54441
E-mail: iog@toolforsystems.com



Defense Information Infrastructure Common Operating Environment

http://diicoe.disa.mil/coe

The Defense Information Infrastructure (DII) Common Operating Environment (COE) is the Defense Information Systems Agency's home page for all DII COE information and operations. It includes documentation, on-line databases that promote information sharing, interoperability and software reuse, Kernel Platform Compliance information, security documentation and tools, links to 28 newsgroups to help exchange information, and much more.



SEI Defense Information Infrastructure Common Operating Environment

www.sei.cmu.edu/activities/str/descriptions/diicoe_body.html

The Software Engineering Institute's Defense Information Infrastructure (DII) Common Operating Environment (COE) is a software technology review. It includes the purpose and origin of DII COE, technical detail, usage considerations, maturity, costs and limitations, and more.

DOD Integrated Digital Environment Web Site

http://ide.dsmc.dsm.mil/default.htm

The DoD Integrated Digital Environment (IDE) Web site was developed to help program managers establish "a data management system and appropriate digital environment that allows every activity involved with the program throughout its total life-cycle to exchange data digitally..." according to a 2 July 1997 memo from Deputy Secretary of Defense White. It includes OSD and Tri-Service Information, IDE resources, information, and pilots.

Open Source Software

www.computerbits.com/gateway/opensource/htm

The Open Source Software site is a listing of open source links to applications, BSD Unices, file archives, Linux, and news and views resources. It is a part of Computer Bits, an Oregon computer information magazine.

TechWeb

www.techweb.com/tech/net_mgt

TechWeb provides a look at news and information created by InformationWeek, InternetWeek, and Network Computing, the three main print and online properties in CMP Media's Business Technology Group. The site contextually links related news, reviews, analysis, opinion, research, and conference offerings from all its sites. Mini-home pages cover each of seven key categories: E-business, Business Applications, Mobile & Wireless, Networking, Security, Network & Systems Management, and Services & Outsourcing.

Global Command and Control System

http://gccs.disa.mil/gccs

The Global Command & Control System (GCCS) is the nation's premier system for the command and control of joint and coalition forces. It incorporates the force planning and readiness assessment applications required by battlefield commanders to effectively plan and execute military operations. Its Common Operational Picture correlates and fuses data from multiple sensors and intelligence sources to provide warfighters the situational awareness needed to be able to act and react decisively. It also provides an extensive suite of integrated office automation, messaging, and collaborative applications. This Defernse Information Systems Agency's Web site features GCCS topics on developer guidance checklist and tools, frequently asked questions, documentation template, reference material, training and more.

EETIMES

www.eetimes.com

EETIMES claims to be the technology site for engineers and technical management. This on-line news center reports on the latest headlines in semiconductors, systems and software, design automation, and technology. It also includes departments, special reports, calendar of events, a product of the week, and more.

GCSS on LAN

I recently attended a pre-solicitation meeting for the acquisition of a new combat support system. The system was to be integrated into the Global Combat Support System (GCSS). The phrase that caught my eye on the slide was "GCSS – Any Box, Anywhere." As the briefer described GCSS as the panacea for all combat support communication, a cynic next to me explained that there was no way that it would be fielded. For every salient point the briefer provided, the cynic had a counter snipe.

I sensed that I had heard this conversation before. The pitch followed by resistance. A counter pitch, met with increased defiance. I was transformed back to my childhood and Dr. Seuss:

I am Uncle Sam
Uncle Sam
Uncle Sam I am

That Uncle Sam-I-am!
That Uncle Sam-I-am!
I do not like that Uncle Sam-I-am!

Do you like GCSS on LAN?

I do not like it, Uncle Sam-I-am.
I do not like GCSS on LAN[1].

Would you like this data share?

I would not like that data share.
I would not like it anywhere.
I do not like GCSS on a LAN.
I do not like it, Uncle Sam-I-am.

Would you like it for the Joint Staff?
Would you like it for a laugh?

I do not like it for the Joint Staff.
I do not like it for a laugh.
I do not like that data share.
I do not like it anywhere.
I do not like GCSS on LAN.
I do not like it, Uncle Sam-I-am.

Would you install it on any box?
Would you use it at Fort Knox?

Not on any box.
Not at Fort Knox.
Not for the Joint Staff.
Not for a laugh.
I would not use that data share.
I would not use it anywhere.
I would not use GCSS on LAN.
I do not like it, Uncle Sam-I-am.

Use it! Use it with NT!
Here it is, it is free.

I would not, could not, with NT.
I would not, could not because it's free.
You let me be.
I do not like it on any box.
I do not like it at Fort Knox.
I do not like it for the Joint Staff.
I do not like it for a laugh.
I do not like that data share.
I do not like it anywhere.
I do not like GCSS on LAN.
I do not like it, Uncle Sam-I-am.

Then train, train - train, train, train!
Could you, would you, if you train?

Not if I train! Not if it's free!
Not with NT! Uncle Sam! Let me be!
I would not, could not, on any box.
I could not, would not, at Fort Knox.
I will not use it for a laugh.
I will not use it for the Joint Staff.
I will not use that data share.
I will not use it anywhere.
I do not like GCSS on LAN.
I do not like it, Uncle Sam-I-am.

Say! On the Web?
Here on the Web!
Would you, could you, on the Web?

I would not, could not, on the Web.

Would you, could you, if in vain?

I would not, could not, if in vain.
Not on the Web. Not if I train.
Not with NT. Not if it's free.
I do not like it, Uncle Sam, you see.
Not for the staff. Not on any box.
Not for a laugh. Not at Fort Knox.
I will not use that data share.
I do not like it anywhere!

You do not like GCSS on LAN?

I do not like it, Uncle Sam-I-am.

Could you, would you, for the CINC[2]?

I would not, could not, for the CINC!
Would you, could you, with a data link?

I could not, would not, with a data link.
I will not, will not, for the CINC.
I will not use it if in vain.
I will not use it if I train.
Not on the Web. Not if it's free!
Not with NT! You let me be!
I do not like it on any box.
I do not like it at Fort Knox.
I will not use it for the Joint Staff.
I will not use it for a laugh.
I do not like that data share.
I do not like it ANYWHERE!
I do not like GCSS on LAN!
I do not like it, Uncle Sam-I-am.

You do not like it. So you say.
Try it! Try it! And you may.
Try it and you may, I say.

Uncle Sam! If you will let me be,
I will try it. You will see.

…

Say! I like GCSS on LAN!
I do! I like it, Uncle Sam-I-am!
And I would use it for the CINC.
I would use this data link.

I won't use it if in vain.
So on the Web I will train.
And with NT, because it's free,
It is so good, so good, you see!

I will install it on any box.
And I will use it at Fort Knox.
And I will use it for the Joint Staff.
And I will use it and not laugh.
And I will use this data share.
Say! I will use it ANYWHERE!

I do so like GCSS on LAN!
Thank you!
Thank you, Uncle Sam-I-am!

This is not exclusive to GCSS, nor does it diminish the hard work of those who have designed and implemented GCSS. It is indicative of all massive, common, global, implementations unique to the Defense Department and the government as a whole. You can replace GCSS in this story with Defense Information Infrastructure (DII) Common Operating Environment (COE), Global Command and Control System (GCCS), Defense Message System (DMS), Defense Information Support Network (DISN), or Ada and the story holds true. The ending may not be as copasetic as Dr. Suess, but the struggle is bona fide.

If DISA would only field a Global Reach Electric Energy Network (GREEN) for Eccentric Global Gambling Sites (EGGS) and Heuristic Award Manipulation (HAM), the parody would come full circle.

[1] *Local Access Network* [2] *Combined Intelligence Center*

— Gary Petersen, Shim Enterprise, Inc.

Sponsored by the Computer Resources Support Improvement Program (CRSIP)

CrossTalk / TISE
7278 4th Street
Bldg. 100
Hill AFB, UT 84056-5205

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737