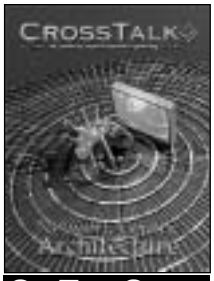
CROSSTALK

August 2003 The Journal of Defense Software Engineering Vol. 16 No. 8

Network-Centric Architecture

Network-Centric Architecture



On the Cover
Cover Design by Kent Bingham.

Software Engineering Technology

Departments

CrossTalk

Sponsor	*Lt. Col. Glenn A. Palmer*
Publisher	*Tracy Stauder*
Associate Publisher	*Elizabeth Starrett*
Managing Editor	*Pamela S. Bowers*
Associate Editor	*Chelene Fortier-Lozancich*
Article Coordinator	*Nicole Kentta*
Creative Services Coordinator	*Janna Kay Jensen*
Phone	(801) 586-0095
Fax	(801) 777-8069
E-mail	crosstalk.staff@hill.af.mil
CrossTalk Online	www.stsc.hill.af.mil/crosstalk
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. 20.

Ogden ALC/MASE
6022 Fir Ave.
Bldg. 1238
Hill AFB, UT 84056-5820

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.

Network-Centric Warfare Brings Increased Combat Power

The theme for this month's CrossTalk is "Network-Centric Architecture." The concept of network-centric warfare has been center stage since the Joint Chiefs of Staff's "Joint Vision 2010" paper was released in July 1996. Since that time, the direction has been reinforced by Joint Vision 2020, and by the Department of Defense's top leadership. Air Force Chief of Staff Gen. John Jumper has been very vocal concerning the need to integrate manned, unmanned, and space systems. "It is through such integration that we achieve the greatest return on our investment in our war-fighting capabilities" says Jumper. "These integration efforts include fully integrating combat, mobility, and space forces into a Joint Synthetic Battlespace …"

The aim of this month's CrossTalk is to explore various aspects of how to achieve this integration of software intensive systems. In the first article, *Designing Highly Available Web-Based Software Systems*, Michael Acton of Lockheed Martin Mission Systems describes the basics of achieving highly available systems. Approaches such as horizontal scaling and cloning are described for producing maintainable, highly available Web applications for hosting on the Global Combat Support System-Air Force (GCSS-AF).

Next, Dr. Malcolm Morrison, Dr. Joel Sherrill, and Ron O'Guin of OAR Corporation, and Deborah A. Butler of the U.S. Army Aviation and Missile Command provide examples of how to achieve mission adaptability in *A Fire Control Architecture for Future Combat Systems*. They describe the need to encapsulate functionality in well-defined software components, to isolate hardware characteristics in personality modules, and to use software architecture to classify components within domains. This work is the foundation for the systems that will support the Army's Future Combat System.

Enterprise Engineering: U.S. Air Force Combat Support Integration by Eric Z. Maass of Lockheed Martin Mission Systems explores the fundamental considerations of developing to an enterprise engineering vision, as well as enterprise application development techniques used on the GCSS-AF. An infrastructure for development, known as the Integration Framework, provides a common set of services and components for applications that join the enterprise. The framework reduces the cost of software development by avoiding reintroduction of common services (e.g., security, messaging, etc.). Success will allow stand-alone combat support systems to integrate in an efficient and secure manner, vastly improving the value of information while reducing the cost of sustainment.

Next, Bradley C. Logan of The Boeing Company, in *Technical Reference Model for Network-Centric Operations*, provides some background on why the shift toward network-centric warfare is needed, and fundamental definitions. He then describes the Strategic Architecture Reference Model (SARM), a technical reference model for network-centric warfare, and how it enables systems-of-systems interoperability. The SARM is a multi-layered model with lower level communications and information layers based on open systems, and higher levels where contractors compete based on their domain expertise. The reference model addresses the need for a guiding framework and products that will allow platforms and systems to become nodes on the Global Information Grid.

With the introduction of network-centric warfare comes increased complexity and the challenge of developing efficient test methods. In *New Spreadsheet Tool Helps Determine Minimal Set of Test Parameter Combinations*, Gregory T. Daich of the Software Technology Support Center, describes a new tool to systematically identify a minimal set of test cases. The approach answers the question: "What is the most effective, smallest set of test configurations that will find the majority of serious parameter interaction defects?"

Network-centric warfare brings increased combat power. Many years ago, it took 1,000 bombs to destroy a target; now that same target can be destroyed by one bomb. What's the difference between those 1,000 bombs and this one bomb? It's the information content of that one bomb. During Operation Enduring Freedom and Operation Iraqi Freedom, we observed how advances in both space-based and unmanned platform technologies allowed persistence over the battlefield, and how advanced sensors provide more precise information. These new capabilities put networks with their ability to quickly disseminate information at the center of military strategy. Our ability to exploit these technologies depends on software professionals like you. We hope this issue will help you in contributing to the design of adaptable, secure, highly available systems.

Glenn A. Palmer

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

Designing Highly Available Web-Based Software Systems

Michael Acton
Lockheed Martin Mission Systems

Highly available Web-based applications require designs that address issues not only of availability, but also of reachability and performance. This article defines high availability and presents approaches for producing maintainable, highly available Web applications. This author cites the Global Combat Support System-Air Force, a network-centric, common technical services-based highly available system as one such example.

To guarantee the quality of service (QoS) deployed warfighters need, combat support systems and applications should be designed to provide near 100 percent availability. Each design activity, which includes usability design, functional design, role-based access control design, and component design, supports the warfighter in some meaningful way. When dealing with the vast technical challenges related to combat support Web application design, it is important not to lose sight of this overarching objective.

The Global Combat Support System-Air Force (GCSS-AF) Air Force Portal, shown in Figure 1, is an example of a Web application that provides dynamic personalized content to warfighters. The Air Force Portal provides both functional and informational capabilities; warfighters can perform their mission using the hosted combat support mission applications, access U.S. Air Force resources and tools, and read about current events. Critical combat support mission applications hosted on the GCSS-AF system, like the Online Vehicle Interactive Management System, Fleet Asset Status, and the Combat Ammunition System, provide excellent examples of the types of Web applications that should be designed for high availability.

One aspect of application development that is often overlooked is ensuring that applications are designed to provide high availability (HA). Proper attention must be given to HA during application design and integration testing to avoid fielding applications that cannot provide 100 percent up time. To illustrate this point, consider the testing phase, which occurs late in development. Why is it that, when compared to functional testing, which usually receives proper attention, HA testing is scarcely considered?

There are many reasons why HA does not receive proper attention during the design phase. One reason is that engineering teams are rushed to meet schedules and do not have the time for proper HA design. Another reason is that many software engineers do not have experience with HA construction and instead focus primarily on functionality. Finally, Web application development is a relatively new paradigm; at least the aspect of developing for a shared enterprise is new.

The remainder of this article defines HA, presents concepts of HA system design, and introduces design approaches for producing maintainable, highly available Web applications.

Figure 1: *The GCSS-AF Air Force Portal at* <https://www.my.af.mil>

HA Described

As viewed, HA is comprised of three key components: *availability*, *reachability*, and *performance*. If any of these components is deficient or fails, then the Web-based system is essentially rendered degraded, or worse, unusable to the warfighters depending on its services.

A system is *available* when all of the necessary system services are up and operating correctly. When the system is available, users can log in through the security subsystem, navigate the Web site, and access and use any of the hosted Web applications for which they have permission. Required services are those that must be operating for a user to receive service. If just one *required* service goes down, the entire system moves into a state of non-availability and becomes incapable of providing *any service* to users. For example, every essential service of the system may be alive except the security subsystem user-authentication service. Without this one crucial service, the system is rendered non-operational. However, system availability is only one element of HA, the system must also be both reachable and performing adequately.

A system is *reachable* when users can access it over the network using a Web browser. Reachability is primarily concerned with assuring robust network service. Problems such as network service outages, improperly configured firewalls and proxies, and router failures will make even an available system unreachable [1].

If the system is both available and reachable, users will receive service, and hopefully with good performance.

For a Web application to be in a highly available state, availability and reachability must be provided in a redundant manner that effectively removes all potential *single points of failure*. Fail-over is a system capability that allows a failed service to automatically be recovered (usually on a secondary or redundant server) in such a manner that the impact on both system processing and users' work is negligible. To ensure fail-over, every server, software service, and network component comprising the system must have at least two independent instances configured. In this manner, users cannot be denied service by any single system component failure.

The *performance* of HA Web applications must, at a minimum, adequately support warfighters in accomplishing their mission. Poor performance hinders user efficiency, creates frustration, and ultimately degrades U.S. Air Force operational capability. Anybody that has ever used a dial-up modem to connect to the Internet has experienced poor performance due to phone line throughput limitations (see Figure 2). T1 network lines provide markedly faster data transfer rates. Imagine warfighters deployed in the desert trying to use a dial-up connection to access critical combat support capabilities.

A monitoring capability for HA systems and applications must be designed as part of the overall solution. Using the system's HA requirements, a comprehensive set of metrics must be established, monitored, and reported to ensure that all aspects of the systems are meeting the required HA thresholds. HA testing is best accomplished by using proactive monitoring tools that actually perform the same actions performed by users. Simple ping-style monitoring tools are not sufficient because they are only capable of checking the system health at the operating system level. These are not capable of monitoring any application-level services. Some important metrics to consider tracking are: overall aggregate system availability (which considers all system services necessary to provide service); overall aggregate system reachability from several locations (i.e., several different bases); and average round-trip time for heavily used capabilities (i.e., a commonly used transaction performed via a Web application).

HA System Design

In order to design HA applications, developers must understand the HA capabilities that the hosting system allows. Two of the most critical design aspects of HA systems are the physical system architecture and session clustering. In order to design HA Web applications, it is necessary to understand how the hosting system implemented these design characteristics.

GCSS-AF has been designed as a highly available system, and affords services that allow Web applications to be hosted on the system in an HA configuration. Availability, reachability, and performance have been painstakingly designed and implemented in the GCSS-AF system to ensure that it performs adequately and that no single point of failure exists. This means that every aspect of the system is immune from any single system component failure.

Every physical server and software service of GCSS-AF is carefully examined during the architecture phase to ensure that at least two independent *silos* are built and configured in such a way that if any one silo were to go down, the remaining silo(s) would pick up the workload. The *intent* is to provide users a system where they do not experience any interruption of service.

Figure 3 illustrates a simplified, tiered, physical-system architecture suitable for hosting HA Web applications (the GCSS-AF architecture is significantly more complex, both in terms of tiers and services). Three Web servers and three Web application servers are interconnected to form a

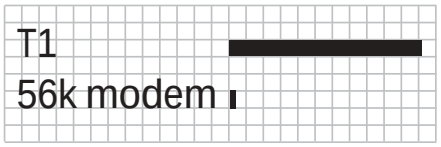


Figure 2: *T1 Network Connections are Capable of Rates of 1.544Mbps Versus a Dial-Up Modem That Is Limited to 56Kbps*

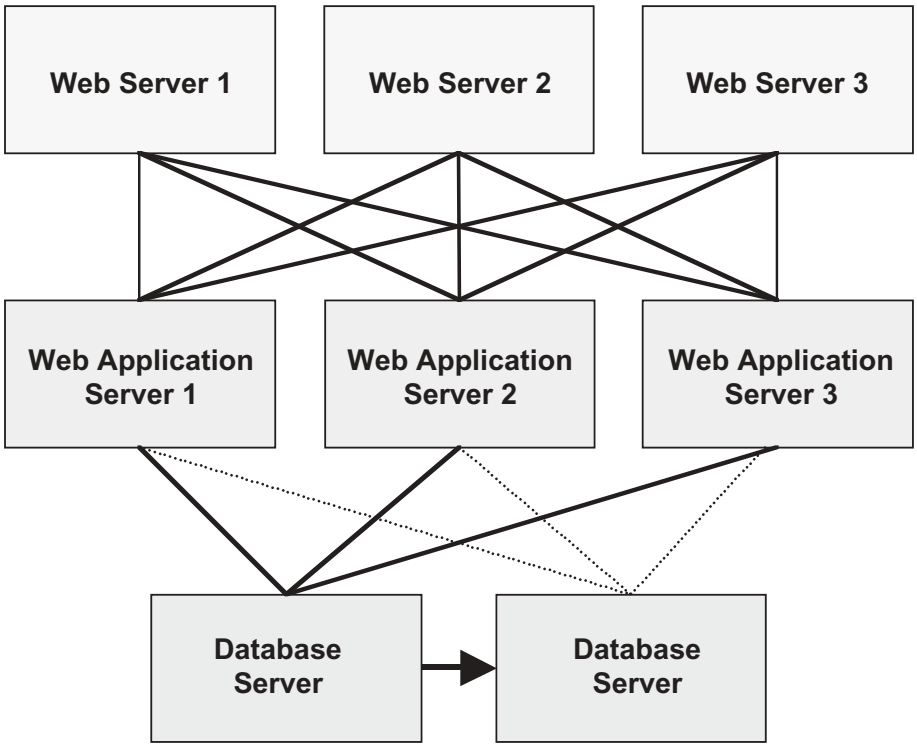
set of redundant services. In the event of any system component failure (i.e., server crash), these networked servers automatically reroute user requests to an available Web application server.

Figure 3 also illustrates simple *fail-over* at the database server tier. Should the primary database server fail, the *hot standby* database takes over and provides service to maintain availability. In this case, the database servers have been configured in a server cluster. The clustering software runs on each of the database servers and can determine when a failure occurs. When this clustering software determines a failure, it promotes the hot standby server to the master database server.

In addition to fail-over, this configuration also provides load balancing of user sessions (at the Web and Web application server tiers). Load balancing coupled with fail-over provides a better operational solution than fail-over alone because it provides for greater scalability and takes advantage of all available capacity, whereas a hot standby idles until needed.

In fact, all GCSS-AF system components have been designed using the HA

Figure 3: *Simplified Tiered System Architecture for Hosting HA Web Application*



principle of eliminating every potential single point of failure. At the network layer, reachability and network performance are provided by modern redundant routers and switches, which ensure multiple access paths to the system. The system is made up of state-of-the-art redundant hardware components that provide exceptional performance and availability (GCSS-AF employs SunFire technology).

At the application layer, performance and availability are provided by redundant software services, which are hosted on different servers to prevent a server failure from impacting software service availability. Additionally, each software component has been designed and configured to support HA optimally. None of this happened by chance; this was all considered in the earliest phases of design.

The second crucial topic relative to HA system design is user session fail-over assurance. A user session is nothing more than a set of related requests and responses between the user and a Web application server. A session is established when a user first accesses the Web application, and is maintained until the user logs off. If for any reason the user session is lost, the Web application server will no longer be able to associate the user with the work they were doing; all of the user's unsaved work will be lost.

The most common causes of user session loss are server and required software service failures (usually in the form of crashes or corruptions). Although the architecture depicted in Figure 3 does provide multiple paths for traversing between tiers, it does not automatically provide user session fail-over. Session fail-over must be designed into the system.

There are essentially three session management approaches used in Web application servers: no session fail-over, database persistent sessions, and memory-to-memory sessions [2]. No session fail-over is unfortunately the most common, and is the default for most Web application servers. In the event of a server failure, all user sessions will be lost along with any unsaved work. These users will be forced to log on again and start over from scratch.

The second approach, database persistent sessions, requires session data to be stored and maintained in a database while the session is alive. If the server fails, then the user's session will be transferred transparently onto another Web application server. However, a performance drawback exists with the database approach – session data must be saved and retrieved from the database in order to maintain the session state.

"Why is it that, when compared to functional testing, which usually receives proper attention, HA testing is scarcely considered?"

The third approach, memory-to-memory sessions, is the best approach. In this approach, session data is replicated among the servers, providing session fail-over and much better performance than the database approach. Because sessions can be transferred among any of the servers in the cluster, both the database and the memory-to-memory session techniques are referred to as *session clustering* approaches. *Session clustering* is required for HA systems like GCSS-AF.

Although session clustering is documented as an open standard in the Java Servlet 2.3 Application Program Interface (API) Specification, clustering does not generally provide fail-over across application servers in different vendor implementations. For example, a Web application hosted on IBM WebSphere cannot fail-over to the Oracle 9iAS and vice-versa. Therefore, session clustering must be addressed separately for each supported application server type (GCSS-AF supports four different Web application servers: IBM WebSphere, Microsoft IIS, Oracle 9iAS, and BroadVision IM; each is independently configured for HA).

Now that some of the important aspects of HA system design have been addressed, we delve into the design considerations for HA Web applications.

Web Application HA Design Considerations

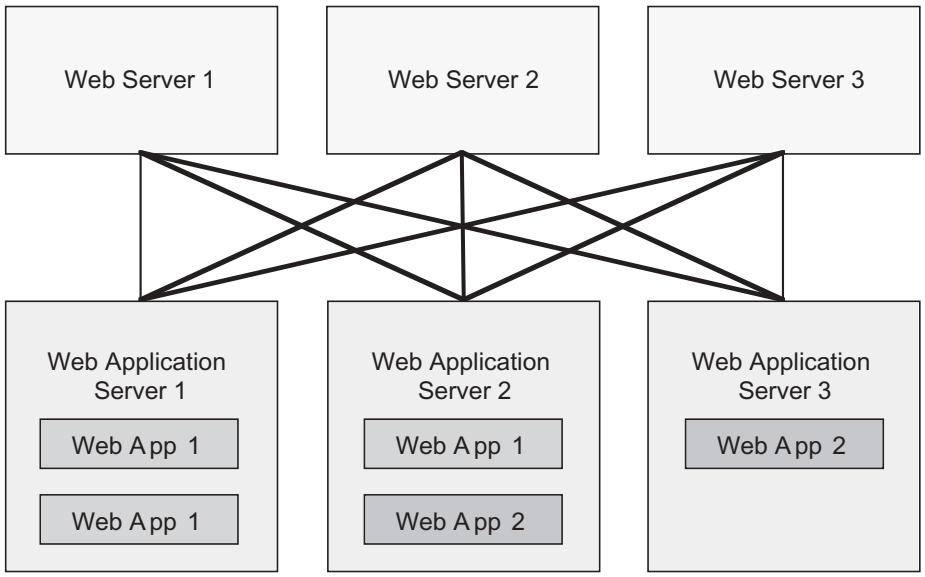
GCSS-AF treats Web applications as software components. Consequently, GCSS-AF hosted Web applications should be designed using approaches similar to the HA system design for software services. The key design elements required for Web application HA include scaling and cloning, treatment of transient data, and wise extension of user sessions with custom objects to provide a survivable storage mechanism for transient data.

Scaling and Cloning

Horizontal scaling means that a Web application has been cloned and is running on at least two independent servers configured to ensure fail-over. Horizontal cloning is required for Web applications to operate in a HA state; it provides fail-over, load balancing, additional user capacity, and enables scalability.

Assuming the application has been designed properly, cloning is a relatively simple task to perform on most Web application servers. For example, on GCSS-AF, it takes about two hours to horizontally clone a combat support mission application. Web application clones can also be placed into a vertical scaling scenario where multiple Web application instances run on one application server.

Figure 4: *Web App 1 Is Scaled Both Horizontally and Vertically. Web App 2 Is Only Scaled Horizontally. Horizontal Scaling Is Required for HA; Vertical Scaling Is Not*



Vertical scaling can provide additional capacity, and does provide server-level fail-over. Figure 4 illustrates Web applications in both horizontal and vertical scaling configurations.

Prior to establishing redundancy through horizontal scaling, a full capacity analysis should be conducted to determine the system resources required to serve the planned user base. The capacity analysis catalogs the total number of users, the average and peak concurrent user load, required application Random-Access Memory per instance, cumulative required hard disk space, and other production-related details to arrive at a physical *fielding profile* for the Web application. The fielding profile depicts how many Web application instances are necessary and lays out precisely which application servers they will reside on.

Additionally, it is important that enough capacity is provided through resource provisioning (Web and application servers, Web application instances, memory, storage, etc.) so that if any one system or application component fails, the *average* user load can still be served. In other words, extra capacity must be allocated by design to handle failures and times of peak usage. The goal of GCSS-AF is to run servers at not more than 40 percent capacity under average load conditions; this allows the necessary excess capacity.

Treatment of Transient Data

Strict separation between layers, as prescribed by the Model-View-Controller (MVC) logical design paradigm [3], results in applications that have business logic and data clearly separated (see Figure 5). This decoupling of layers greatly reduces complexity, fosters code reuse, enables flexibility, and simplifies maintainability throughout the life cycle. In short, MVC provides the necessary logical design foundation for developing maintainable and HA applications.

MVC-designed applications should not manipulate database data directly using Java Database Connectivity (JDBC); instead, they should take a pure object-oriented approach and interact with data using objects [4]. Data objects are implemented as Entity Beans or simple JavaBeans, and are buffered by the data access layer, which directly communicates with the database via JDBC.

Using the MVC logical design approach results in Web applications that are suitable for running in a HA mode because they scale nicely (both horizontally and vertically). However, using MVC does not prevent developers from making HA design mistakes. For instance, MVC does not preclude the use of transient session data. Transient session data can best be viewed as the user's unsaved work on the Web application server. If the server fails, all transient data is lost, just like when a workstation user experiences a power outage with unsaved work. In the event of a failure, HA-enabled Web applications will fail-over to another server, however, all transient data will be lost. HA Web applications must be designed in such a fashion that a session fail-over will not result in the loss of *any* of the user's work. Specifically in Java 2 Enterprise Edition, the servlets, JavaBeans, Java classes, and Enterprise JavaBeans (EJBs) can contain transient data [5].

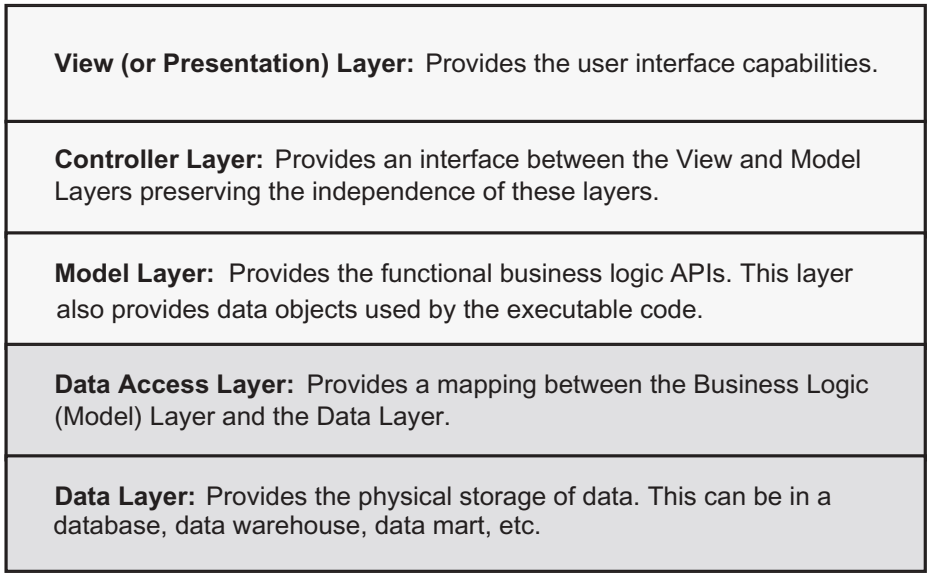


Figure 5: *Model-View-Controller Logical Tiered Architecture*

Session EJBs illustrate the pitfalls of using transient data in HA Web applications. Session EJBs can be either stateful or stateless. Stateful session beans maintain transient state information (in-memory data) that is used to serve or converse with the user. Stateful sessions are valuable because they allow data caching, which can dramatically improve performance. However, they do have a dark side. Should the application server fail, all of this transient data is permanently lost. For example, consider a warfighter placing an order for communications equipment (using a combat support mission Web application similar to Amazon.com). As the warfighter places items into the shopping cart, the stateful session bean keeps track of the requested items in-memory. The EJB now has state, namely this in-memory list of items. Completing the order and checking out depends on the in-memory state information remaining available. The problem with this approach for HA Web applications is that when a server fails, this state information is lost and cannot be recovered. The user will be disrupted, and be forced to start over [6].

Properly designed HA Web applications can prevent users from experiencing transient data loss by employing stateless objects. Stateless session beans maintain no in-memory state data that could be lost in the event of a server failure. Each method in a stateless session EJB executes independently and does not rely on or store any in-memory state data. Again, consider the warfighter shopping cart example; since a stateless session bean does not maintain state data, the bean must add the warfighter's selected items to a persistent object (which moves the data into a database table via the Data Access Layer, see Figure 5). At checkout, the selected items will be retrieved from this table and the transaction will be completed. In the event of a failure, the clustered session would fail-over to another application server, and upon checkout the new application server would simply read the list of selected items from a persistent object (which pulls the data from the database table) and complete the order. Because of this resiliency, HA Web applications should only use stateless session beans. This treatment of transient data applies to servlets and regular JavaBeans as well.

Customizing the Session Object

If a Web application has the need to store information about a user in addition to what the user session object provides, a good approach is to extend the session by adding custom objects. Returning to our

COMING EVENTS

September 8-12
International Conference on Practical Testing Techniques
Minneapolis, MN
www.psqtconference.com/2003north

September 15-18
Software Development Best Practices
Boston, MA
www.sdexpo.com/2003/east

September 14-19
International Function Point Users Group Annual Conference
Scottsdale, AZ
www.ifpug.org/conferences/annual.htm

September 22-25
AUTOTESTCON 2003
Anaheim, CA
www.autotestcon.com

September 24-26
International Conference on Visual Languages and Computing
Miami, FL
www.vlc03.cs.ucla.edu

October 15-18
Richard Tapia Diversity in Computing Conference
Atlanta, GA
www.ncsa.uiuc.edu/Conferences/Tapia2003

October 20-24
Quality Assurance Joint Conference on Compressing Software Development Time
Baltimore, MD
www.qaiusa.com

November 18-21
International Conference on Software Process Improvement
Washington, DC
www.software-process-institute.com

April 19-22, 2004
Software Technology Conference 2004

Salt Lake City, UT
www.stc-online.org

warfighter shopping cart example, we could extend the user session object by adding an object called SelectedItems (which is a Java ArrayList of EquipmentItems). The warfighter's selected items can now be stored in the new SelectedItems object of the user's session. To process the checkout, these items are simply retrieved from this in-memory object (assuming memory-to-memory session clustering is used). In the event of a server failure, these objects will be treated as part of the user's session and will be subject to session clustering fail-over previously discussed. Thus, a server failure will cause a complete fail-over of the user's session and *transient data* without the user ever experiencing any interruption in service. This approach also provides a gain in performance because the SelectedItems are stored in memory instead of the database. This is a good approach for a relatively small amount of data. Large amounts of data should not be stored with the user session object [2].

Conclusion

Developing HA systems and applications is not about technology. It is about meeting the needs of the warfighter. To serve the warfighter, Web applications should be designed up-front to provide HA, which can be viewed as a composition of three components: availability, reachability, and performance. Each of these HA elements must be designed into the hosting system as well as the Web application. HA systems must ensure that the architecture *designs away* all single points of failure. The most important consideration for HA Web applications is horizontal scaling. Other important aspects of HA design include strict separation between layers (using MVC), the use of session clustering and fail-over, the treatment of transient data, and extending the session object with custom objects.◆

References

1. Tanenbaum, A. Computer Networks. 3rd ed. New Jersey: Prentice Hall PTR, 1996.
2. IBM WebSphere Version Information Center: Session Management Support <http://publib7b.boulder.ibm.com/wasinfo1>.
3. Alur, D. et al. Core J2EE Patterns: Best Practices and Design Strategies. New Jersey: Prentice Hall PTR, 2001.
4. Sun MicroSystems: Java Data Object API Specification <http://java.sun.com/products/jdo>.
5. Hall, M. Core Servlets and JavaServer Pages. New Jersey: Prentice Hall PTR, 2000.
6. Monson-Haefel, R. Enterprise Java Beans. 2nd ed. Sebastopol, CA: O'Reilly, 2000.

® Capability Maturity Model is registered in the U.S. Patent and Trademark Office.

Additional Reading

1. Alberts, David S. Network Centric Warfare: Developing and Leveraging Information Superiority. 2nd ed revised. CCRP Publication Series, 2000.
2. Java and J2EE Timeline <http://java.sun.com/features/2000/06/time-line.html>.
3. Martin, J. "On Service Level Agreements for IP Networks." Proc. of the IEEE Infocom 2002 Conference <http://www.ieeeinfocom.org/2002/papers/455.pdf>.
4. Pressman, R. Software Engineering, A Practitioner's Approach. 4th ed. New York: McGraw-Hill, 1997.
5. Stevens, W. UNIX Network Programming. 2nd ed. New Jersey: Prentice Hall PTR, 1998.
6. Wang, Z. Internet QoS: Architectures and Mechanisms for Quality of Service. San Francisco: Morgan Kaufmann, 2001.

About the Author

Michael Acton is a software systems engineer with Lockheed Martin Mission Systems, a Capability Maturity Model® Level 5 organization. As the Global Combat Support System-Air Force (GCSS-AF) Operations and Support chief engineer, he is responsible for leading multiple engineering teams that provide the services necessary to operate the system, provide Level 2 help-desk support, install new capabilities, and integrate Java 2 Enterprise Edition-based mission applications. He was recognized as a 2002 GCSS-AF Program Top Contributor. Acton is a doctorate candidate at Auburn University.

Lockheed Martin
Mission Systems
4520 Executive Park Drive
Montgomery, AL 36116
Phone: (334) 416-6029
Fax: (334) 273-5560
E-mail: michael.acton@gunter.af.mil
michael.acton@lmco.com

A Fire Control Architecture for Future Combat Systems

Dr. Malcolm Morrison, Dr. Joel Sherrill, and Ron O'Guin
OAR Corporation

Deborah A. Butler
U.S. Army Aviation and Missile Command

The Future Combat Systems (FCS) program is developing a versatile, reconfigurable system of systems capable of performing a wide range of missions for the U.S. Army. In support of the FCS, the Fire Control-Node Engagement Technology (FC-NET) program is developing a versatile, modular fire control software architecture capable of satisfying the flexibility and rapid mission reconfiguration requirements of the FCS. The FC-NET software architecture achieves its flexibility through domain-centric software components that encapsulate weapon-specific hardware devices and algorithms. This article illustrates the structure and organization of the architecture using the Munitions domain as an example. A side benefit of developing and testing the FC-NET architecture is the production of reusable artifacts that offer potential cost and schedule savings on future FCS evolution and system integration efforts.

The U.S. Army's Future Combat Systems (FCS) program is developing a network-centric ensemble of systems capable of performing a range of missions, from warfighting to peacekeeping. Capabilities envisioned for FCS include direct and indirect fires, air defense, reconnaissance and surveillance, transport, and resupply [1]. Current FCS vehicle concepts range from unmanned aerial vehicles to manned 20-ton wheeled or tracked vehicles to small robots that weigh only a few pounds [2].

Developing systems that bring the FCS vision to reality requires the collaborative efforts of numerous communities. Architecture descriptions of the FCS will enable those communities to quickly and efficiently engineer, procure, and deploy advanced systems. The Fire Control-Node Engagement Technology (FC-NET) software architecture is being developed by the U.S. Army Aviation and Missile Research, Development, and Engineering Center as a foundation for fire control systems that support FCS. The essential characteristic of FCS is mission adaptability, and FC-NET provides a fire control software architecture that is equally adaptable.

Fire control in general encompasses all operations required to apply fire on a target [3]. Fire control systems are categorized as either tactical or technical. Tactical fire control systems are Command, Control, and Intelligence systems that focus on the planning and evaluation aspects of fire control. Tactical fire control systems are responsible for such activities as identifying targets based on data from multiple sources, prioritizing targets, assigning specific weapons for use against specific targets, and assessing damage after engagements.

Technical fire control systems are normally embedded in a weapon system and focus on the computational and mechanical operations required for that weapon system to hit a specific target with a specific munition. Technical fire control systems are responsible for interacting with tactical fire control systems to obtain information about targets. This information may be used to direct fire or to further refine firing solutions using organic sensors under the control of the technical fire control system.

Once sensors have acquired a potential target, the technical fire control system assists in tracking the target until a decision is made to engage and fire upon the target. Platforms and vehicles provide mobility for aiming weapons and sensors. Technical fire control systems augment the soldier's capability, enabling the soldier to fire on more targets both more quickly and more accurately. Figure 1 illustrates this role of technical fire control systems as force multipliers for the individual soldier.

FC-NET is a technical fire control software architecture. The architecture's flexibility ensures that FC-NET-based technical fire control systems can readily interact with tactical fire control systems and fully exploit the information that the tactical systems provide. However, the architecture's structure emphasizes the technical actions required to place munitions on targets.

As used in the remainder of this article, the term *fire control* should be interpreted as meaning *technical* fire control.

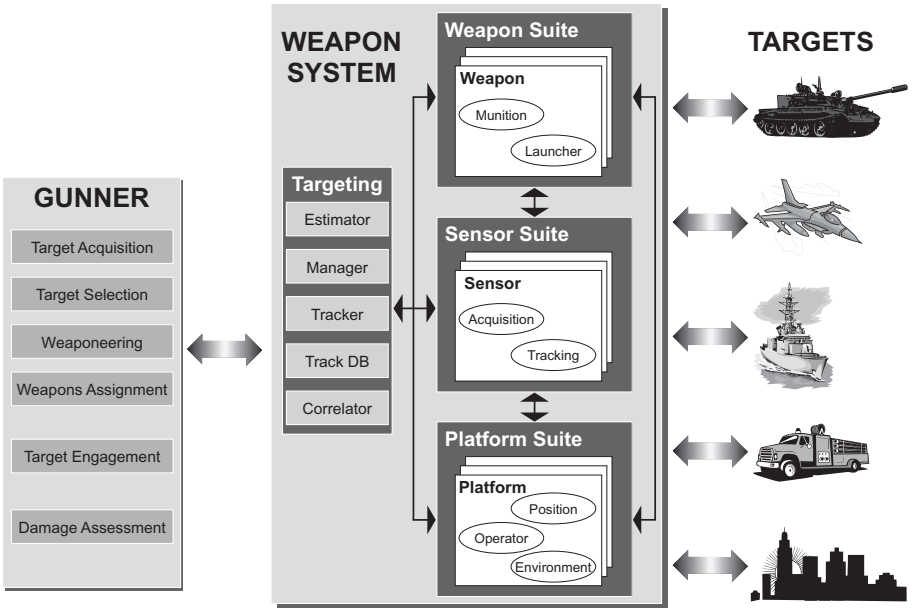
The FC-NET Architecture

FC-NET captures technical fire control functionality in a modular software architecture that provides a *plug-and-fight* capability for FCS. The following sections provide a brief description of how the architecture was developed from domain models, how it is represented as a set of software components, and how it is expected to evolve.

Domain Modeling

A domain is defined by systems that perform similar missions. For example, bullets, rockets, and missiles all belong to the Munition domain. A domain model is an abstraction of the systems within a

Figure 1: *Technical Fire Control Is a Force Multiplier*



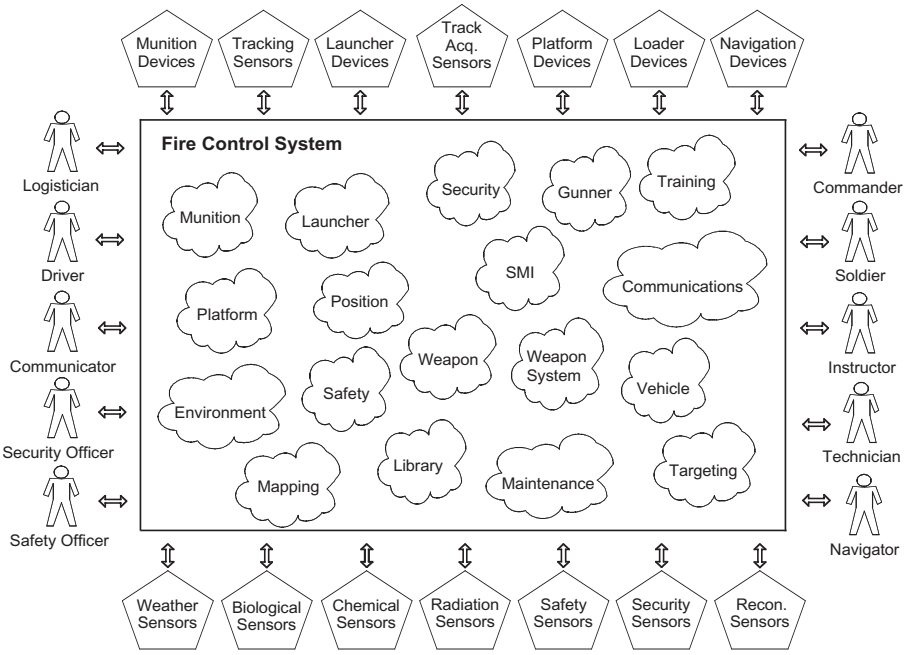


Figure 2: *Fire Control Domain and Interactions*

domain. A domain model is built for the purpose of understanding the domain's requirements, information, and processes. The top-level domain for the FC-NET architecture is military fire control systems. Therefore, the FC-NET architecture domain model is a model of the requirements, information, and processes associated with military fire control systems.

The FC-NET domain modeling activity examined the information required by a fire control system to execute a fire mission. Since fire control systems are composed of a variety of sensors and electromechanical devices, it was critical to classify the devices based upon the information they provide and the roles they play. This focus on devices helped draw a strong boundary between the problem space addressed by the FC-NET architecture and that of subsystems with which a fire control system must interact. Examples of such subsystems include battlefield management, avionics control, vehicle control, and user interfaces.

Figure 2 identifies the roles and devices that interact with the top-level fire control system domain. Within the fire control system domain, a number of other lower-level domains (or sub-domains) have been identified such as Weapon and Munition. Each of these lower-level domains represents a significant area of functionality and serves a coherent role in a military fire control system. Each lower-level domain is further decomposed into units called components.

Components

A component is a logically coherent, distributable unit of software composition. The behavior of a domain emerges from the interactions among its components and between its components and the external environment. The following criteria were used to determine if an area of functionality should be encapsulated into a single component:

- **Logically Coherent**. Is there a collection of functional capabilities that should be grouped because of interdependencies?
- **Distributable**. Could this functionality be distributed?
- **Vendor Provided**. Could a vendor provide this functionality as a standard component?
- **Special Expertise**. Does a subject-matter expert normally write this functionality?
- **Updateable**. Could updates to this functionality occur independently of updates in other areas of functionality?
- **Optional**. Is this functionality only required in some systems?

The following example makes using these criteria clearer. A computationally intensive algorithm for target recognition might execute on a separate processor from software controlling the sensor devices. In this case, it would be desirable to be able to make sensor control and target recognition separate components to provide flexibility to the system designer in assigning these to separate processors. It is also likely that a subject-matter expert would provide the target recognition algorithm. The subject matter expert would normally not be responsible for the software interfacing with sensor devices. Ideally, the manufacturer of the sensor device would provide this functionality. Finally, it is reasonable to expect improvements to both target recognition algorithms and sensor devices. It is likely that these would occur independently of one another.

In this example, the criteria suggest that target recognition capabilities and sensor device control be placed in separate components. The separation of functionality illustrated by this example is supported by FC-NET's adherence to an open system model with standard component interface definitions.

The FC-NET architecture identifies components within each of the lower-level fire control domains. The specifications for individual components are defined in terms of the attributes they possess, the services they offer, and the asynchronous notifications they may generate.

Attributes may be thought of as data elements. However, they are only accessible to clients of the enclosing component via services that access or modify the attributes. These accessing services are known informally as *get* and *set* and implicitly exist for every attribute of a component. In multi-threaded environments, accessing services are assumed to be thread-safe and provide atomic access to attributes.

Services define operations that can be performed by the component. Services may modify the values of attributes and may invoke the services of other components. Services may perform computations or effect changes in the external environment. For example, a fire control system must provide an Aim service that results in parts of a weapon system being physically repositioned so that munitions can be fired at targets.

Notifications are asynchronous messages generated by a component when an event of interest occurs. Other components subscribe to receive those notifications in which they have an interest. For example, the Department of Defense defines a hang fire as a non-desired delay in the functioning of a firing system [3]. A hang fire occurs when a weapon is fired but the munition does not physically leave its launcher. A notification is used to pass knowledge of this event to other components that need to know when a hang fire occurs.

For ease of modeling, the FC-NET domain model classifies components within each domain into one of five categories: functional controls, knowledge stores, device groups, algorithm containers, and façades.

Functional controls encapsulate sets of

functionality that have a similar purpose and perform a specific role within a fire control system. Since functional control components have specific roles, they are named according to their role. Functional controls always have services but sometimes do not have attributes or notifications.

Knowledge stores encapsulate groupings of related information. Each knowledge store component provides controlled access to its information, even in the face of concurrency. In general, knowledge stores are global repositories of system data. All knowledge stores are named according to the type of data they encapsulate. Knowledge stores tend to have attributes and implicit accessing services for reading and modifying the attributes, but rarely have any explicit services or notifications.

Device groups encapsulate collections of sensors or controllers that are used for similar functions. Specific devices are not defined within a device group. Since device groups represent hardware components, they are named according to the generic hardware that they represent. Device groups tend to have attributes, services, and relatively many notifications.

Algorithm containers encapsulate computations that are highly complex, likely to be system-specific, and typically written by subject-matter experts. In the fire control domain, examples include weapon/target pairing, target identification, and ballistic computations. Algorithm containers tend to contain very few services and no attributes or notifications. Algorithm containers normally operate in demand mode – executing only in response to requests from other components or as a result of system events.

Façades provide high-level interfaces that make complex subsystems easier to use by encapsulating the data and functionality of the subsystems. A façade is especially useful when the subsystem is highly complex, adheres to another domain model, or is produced by an outside organization. Although façades tend to have moderately complex interfaces, the implementation of a façade usually consists of straightforward mappings between subsystem and façade attributes, services, and notifications. An example of a façade in the FC-NET architecture is the Vehicle component. The Vehicle component is responsible for presenting the fire control system with a unified interface to all the subsystems that are part of (or are attached to) the vehicle on which the weapons are mounted. Common examples of such vehicle-mounted subsystems include power plant control and monitoring systems, speed and position sensors, meteorological sensors, and nuclear, biological, and chemical detection systems.

Examples of these component categories are provided in a later section of this article that presents a decomposition of the Munition domain.

Architecting for the Future

FCS is being designed as a highly adaptable and flexible fighting platform, but it is still largely conceptual. The modular, component-based modeling approach described earlier provides the architectural adaptability and flexibility required if FC-NET is to support an evolving FCS. Although the exact physical nature of FCS is not known with certainty at present, initial concept definitions hint at possible configurations.

Possible FCS Configurations

For this discussion, a *weapon* is defined as the composition of a platform, one or more launchers, and one or more munitions. Weapons may be fixed or mobile. If mobile, the weapon platform must be mounted on some type of vehicle. The overall movement capability of a weapon depends on both the movement capabilities of a vehicle and the articulation capabilities of a platform. Likewise, sensors used to acquire and track targets require movement capabilities similar to the weapons they help control. Conceptual depictions of FCS to date embody, at minimum, the three different weapon and vehicle configurations shown in Figure 3.

Figure 3(a) reflects the simplest case where a weapon and its sensors are boresighted along the same line. In this configuration, all sensors and weapons effectively are aimed at the same point and moved at the same time. Thus, aiming at and tracking a target with a sensor results in the weapon aiming at and tracking that same target.

A step up on the complexity scale has each sensor or weapon located on a single vehicle but capable of independent movement. This configuration is shown in Figure 3(b). When weapons and sensors are mounted on the same vehicle, their actions must be coordinated to avoid interference and ensure proper aiming and alignment.

The most complex and most flexible configuration allows sensors and weapons to have completely independent movement. This situation is depicted in Figure 3(c) and reflects the growing utilization of unmanned ground and aerial vehicles. Coordination of sensors and weapons is still an issue, but the coordination may need to be effected across significant distances.

Personality Modules

Isolation of the fire control system from the underlying hardware configuration is handled in FC-NET by using Personality Modules (PMs). Similar to the device drivers used by popular operating systems, PMs encapsulate device-specific hardware characteristics. A PM implements an architecture-defined interface to the fire control system. The PM translates abstract fire control commands into device-specific commands understood by the attached device. Figure 4 (see next page) presents an example.

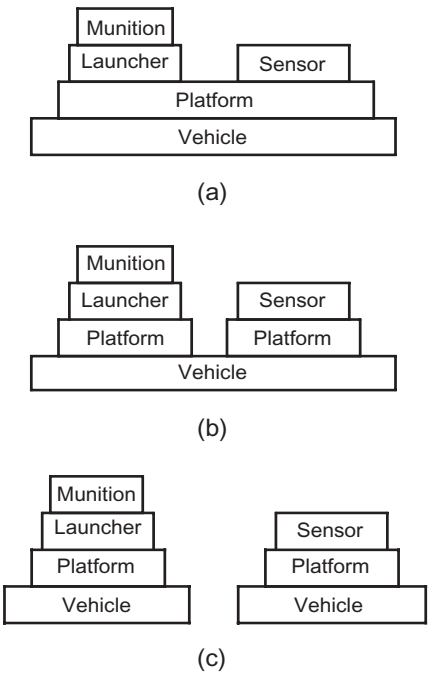
The fire control system sends a *Fire* command to the software component that controls a missile launcher. The software component passes the fire command to the launcher hardware through a launcher-specific PM. The PM translates the fire command into a sequence of relay activation commands that control electrical signals to the launcher at the connector pin level.

Fire Control Foundation Classes

Although weapon system physical configurations may vary widely, the fundamental operations required to place fire on targets remain relatively constant. Any fire control system must acquire and track targets, compute firing solutions, and deliver munitions against targets. The FC-NET exploits this functional consistency wherever possible to support the FCS in all of its anticipated weapon configurations.

The modular, domain-oriented structure of the FC-NET architecture promotes commonality at an abstract level. The architecture is designed to encourage and

Figure 3: *Possible FCS Configurations*



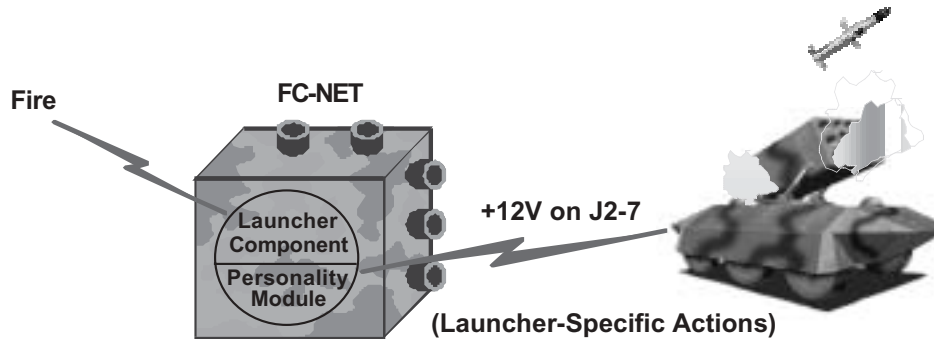
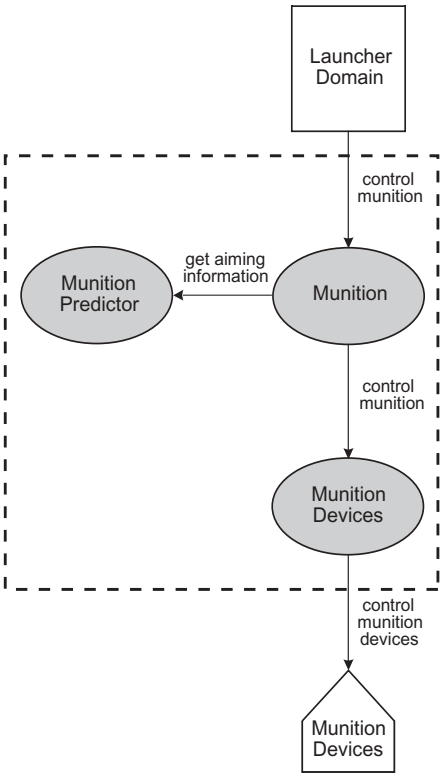


Figure 4: *Personality Module Example*

accommodate reuse. The architecture itself with its defined interfaces and services can be reused, as can implementations of components written in conformance with the architecture. As the FC-NET architecture is developed, a reference implementation of the architecture and a set of fire control foundation classes will be created along the lines of Microsoft Foundation Classes (MFC) [4]. An example of a potential fire control foundation class is geospatial position. Targets have positions, weapons have positions, and munitions have positions during their flight from weapon to target. A reusable position foundation class has been defined that provides position-related services such as transformations between coordinate systems. The position foundation class can be instantiated or extended by any fire control system implementer.

Just as the MFC are tightly coupled to Microsoft's Document-View application architecture, the FC-NET fire control foundation classes will be tightly coupled to the FC-NET fire control architecture. Unlike the MFC, however, the FC-NET will be a non-proprietary open system portable to a variety of operating systems and central processing unit families.

Figure 5: *FC-NET Munition Domain Interaction*



Sample FC-NET Architecture Domain

Concepts discussed in earlier sections are illustrated by examining the part of the FC-NET architecture that addresses munitions. The FC-NET view of the Munition domain appears in Figure 5. The figure shows that the Munition domain contains three interacting software components: Munition, Munition Predictor, and Munition Devices. These components interact in turn with munition-related hardware devices and with other software components in the Launcher domain. Similarly, the Launcher domain is also composed of interacting software components that interface to hardware devices and to software components in yet other domains in the architecture.

Munition is a functional control component that is responsible for munition targeting and launching functions. Example services provided by this component include activating, arming, and launching a munition. The Munition component is responsible for sending notifications of events that occur during the course of its operations. For example, this component generates notifications when a munition is launched or stowed. These particular notifications normally originate in the Munition Devices component and the Munition merely propagates the notifications out to other components in the system. The Munition component also maintains information about munition attributes. This information includes basic characteristics, configuration information, and dynamic state. Example attributes include munition type and model number. Guided munitions may have attributes for laser designator code or waypoint list.

Munition Devices is a device group component that provides an interface for controlling the devices and sensors associated with a munition. It is used by the Munition component to access a munition's physical hardware. This component provides many of the same abstract services as the Munition component, such as aim, arm, and launch, but implements these services at a lower, hardware-aware level. The Munition Devices component generates the same notifications as the Munition component but does so at a lower level.

Munition Predictor is an algorithm container component that provides an interface for determining the necessary aiming conditions required to ensure that the munition's kill zone intersects the target. The Munition Predictor is used by the Munition component to compute the aim point for a munition. Like all algorithm containers, the Munition Predictor has no attributes and generates no notifications. The only services provided are computation of munition flight time, aim angle, and lead.

Design Trade-offs

Every software design involves trade-offs, and the FC-NET is no different in this regard. The FC-NET fire control architecture reflects the hardware structure and operational environment of modern weapon systems. The architecture is highly modular and assumes a hierarchical control model among components. The current version of the architecture defines 47 components, along with some additional data types and support services that collectively capture the behavior of 18 domains.

At first glance, this breadth and level of decomposition might seem excessive. Although this observation might hold true for many current fire control systems, the FC-NET was not designed for current fire control system. The FC-NET was designed for future fire control systems. In particular, it was designed for the FCS fire control system. The architecture was driven by the need to support the highly adaptable and reconfigurable weapon system that the FCS will be. Of particular concern is the potential for future weapon systems to be composed from cooperating, autonomous robots or physically distributed subsystems.

There are costs associated with such a modular and hierarchical architecture. Architectural simplicity may be reduced, as may implementation efficiency. The munition example presented earlier showed three independent, interacting components that together provide the necessary domain behavior. Other device-centered domains such as platforms and launchers also have

separate functional control and device group components. For simple devices, one could argue that this is design overkill. Given a dumb launcher device that does nothing more than transfer firing voltages to its mounted missiles, multiple components could introduce needless inefficiencies. Performance penalties can be incurred due to cross-component communications. Added complexity and risk can be incurred if it is necessary to locally cache data needed by multiple components.

Contrast this dumb launcher example with that of an intelligent launcher device with an autoloader, both mounted on an autonomous robot. Cross-component communications and local data caching become inevitable consequences of the hardware configuration. The architectural complexity that seemed excessive for the first example actually provides a smoother implementation path for the second.

Although complexity and inefficiency are legitimate concerns, current trends in processing speed and communication bandwidth ameliorate these disadvantages. Computer hardware gets faster and cheaper every year. The IBM PowerPC 750FX is a 32-bit processor that operates at speeds up to one gigahertz and is representative of high-end processors being used in new embedded systems [5]. Powerful processors can be teamed with Flash memory to provide large amounts of primary storage and virtual file systems. Advanced processors such as the PowerPC family have dramatically increased the capabilities of recent embedded systems, and the trend toward more powerful hardware is expected to continue.

In the case of distributed fire control systems, communications among physically independent components can have a major impact on overall system efficiency. The speed and reliability of network data transmission is almost always less than the speed and reliability of data transmission within a single processing unit. It is anticipated that distributed communication architectures such as Real-Time CORBA [6] will continue to mature and grow in reliability, performance, and usability to the point where communications issues cease to be a major performance concern for the architecture.

In light of these disadvantages, what advantages does the FC-NET architecture provide? Although we believe that there are potentially many, this article focuses on only three: adaptability, interchangeability, and vendor-independence.

An adaptable architecture allows fire control systems to be easily modified, extended, or reconfigured in the face of changing requirements. Strategic defense policy formulation requires long-term planning. Long term in this context involves time horizons of 10 or more years as exemplified by Joint Vision 2010 [7] and Joint Vision 2020 [8]. Unfortunately, it is impossible to predict with any certainty what will happen to hardware and software technologies over the same period. It is important to design any new fire control system in such a manner that new technologies can be readily incorporated as they become available. The highly modular character of the FC-NET architecture provides numerous locations where new technologies can be inserted into the system with minimal impact on other components in the fire control system.

An architecture that provides easy interchangeability of components allows fire control systems to be readily modified, extended, or reconfigured to better meet current requirements. Individual components of the fire control system can be swapped out to incorporate improved algorithms, more efficient component implementations, or more sophisticated decision aids that enhance the weapon system's ability to engage targets. As commercial technologies mature and are adopted by the Department of Defense, new components that utilize these technologies can replace older components based on more expensive military technologies.

For example, if a system moves from MIL-STD-1553B-based communications to Transmission Control Protocol/Internet Protocol (TCP/IP), the FC-NET architecture ensures that this change only impacts boundary components involved with hardware device communications. Most importantly, component interchangeability supports the plug-and-fight capability of the FCS to host different combinations of weapons at different times. For example, a FCS might be quickly reconfigured by exchanging a non-line-of-sight gun for a complement of Netfires missiles and a remote armed reconnaissance robot. The FC-NET is designed to accommodate distributed fire control systems, where fire control software components physically reside in weapon hardware. The required software components are then automatically incorporated into the fire control system when the weapon hardware is inserted into the weapon system.

An architecture that provides vendor-independence allows fire control systems to be composed of components created by different vendors. Vendors can work independently to produce fire control components that interoperate with components provided by other vendors or the government, using whatever in-house expertise or methodologies provide their competitive advantage. Since every component developer writes to defined component interfaces, integration costs are reduced and subcontracting becomes an effective means of incorporating best-of-breed technology into a fire control system.

Program Status

An initial version of the FC-NET software architecture has been produced [9]. The FC-NET program is in the first phase of a five-phase 50-month effort that will refine the software architecture through application to four different weapon systems constructed from five different weapons.

Each weapon system will be composed of two or more weapons and will consist of some combination of real and simulated hardware. The weapons selected for these systems will be representative of the types of weapons the FCS expects to mount. Example weapons include the Low Cost Precision Kill Missile, Common Missile, Compact Kinetic Energy Missile, and an Objective Crew Serve Weapon. Each weapon system will feature realistic gunner interaction by utilizing an integrated crew station provided by the Army's Tank and Automotive Research, Development, and Engineering Center.

The first phase of the program culminated in a June 2003 demonstration of a weapon system that operates in a simulated environment. It is expected that the FC-NET reference implementation and fire control foundation classes described earlier will be produced as by-products of implementing the different weapon systems developed during the course of the program. One intent of the demonstration projects was to expose opportunities for improving the current architecture. Early implementation experience has already resulted in many minor changes to component interfaces. More substantial changes to the architecture will be considered at the conclusion of each program phase.

Conclusion

The FC-NET fire control software architecture provides the flexibility needed to support the FCS and other new and evolving weapon systems. This flexibility is achieved through the encapsulation of functionality in well-defined software

components and the isolation of hardware characteristics in PMs. Fire control foundation classes and a reference implementation of the architecture will be developed in conjunction with this program. System integrators can exploit these reusable artifacts to achieve cost and schedule economies when developing fire control systems for new configurations of the FCS.◆

References

1. United States. Defense Advanced Research Projects Agency. "DARPA FCS Overview." Washington: DARPA, 26 Mar. 2002 <www.darpa.mil/tto/programs/fcs.html>.
2. McElwee, J., and J. Gully. "Future Combat Systems: Partnering for Rapid Innovation and Transformation." Boeing Integrated Defense Systems, 4 Apr. 2002 <www.boeing.com/defense space/ic/fcs/bia/mcelwee.zip>.
3. United States. Joint Force. DoD Dictionary of Military and Associated Terms, Joint Publication 1-02. Washington: Director for Operational Plans and Joint Force Development (J-7), 12 Apr. 2001 (as amended through 15 Oct. 2001) <www.dtic.mil/doctrine/jel/doddict>.
4. Prosise, J. Programming Windows With MFC. Redmond, WA: Microsoft Press, 1999.
5. IBM Microelectronics Division. "PowerPC 750FX Product Brief." IBM Corporation, Apr. 2002 <www.3.ibm.com/chips/techlib/techlib.nsf/products/PowerPC_750XF_Microprocessor>.
6. Schmidt, D. C., and F. Kuhns. "An Overview of the Real-Time CORBA Specification." IEEE Computer. June 2000 <www.cs.wustl.edu/~schmidt/PDF/orc.pdf>.
7. United States. Joint Chiefs of Staff. Joint Vision 2010. Fort Belvoir, VA: Defense Technical Information Center, July 1996 <www.dtic.mil/jv2010/jv2010.pdf>.
8. United States. Joint Chiefs of Staff. Joint Vision 2020. Fort Belvoir, VA: Defense Technical Information Center, June 2000 <www.dtic.mil/jv2020/jvpub2.htm>.
9. U.S. Army Aviation and Missile Research, Development, and Engineering Center. FC-NET Architecture Description, v. 1.1. U.S. Army AMCOM, June 2002.

About the Authors

Joel Sherrill, Ph.D., is director of Research and Development for OAR Corporation with 15 years experience in the design, development, and fielding of real-time embedded applications in a variety of military, commercial, and research domains. As a principal author and current maintainer of the open-source real-time operating system RTEMS, he has been deeply involved in numerous RTEMS-related efforts including the GNAT/RTEMS validation. Sherrill is a founding member of the Steering Committee for the Free Software Foundation's GNU Compiler Collection.

OAR Corporation
4910-L Corporate Drive
Huntsville, AL 35805
Phone: (256) 722-9985
Fax: (256) 722-0985
E-mail: joel.sherrill@oarcorp.com

Ron O'Guin is executive vice president for OAR Corporation with 25 years experience in the development of real-time operating systems, real-time applications, visual simulations, and weapon systems trainers. As a principal author of the open-source real-time operating system RTEMS, he has been deeply involved in numerous RTEMS-related development efforts. O'Guin has an extensive background in missile system research and is a principal developer of the Fire Control-Node Engagement Technology (FC-NET) Technical Fire Control Architecture. He currently serves as the software manager for the FC-NET STO Program.

OAR Corporation
4910-L Corporate Drive
Huntsville, AL 35805
Phone: (256) 722-9985
Fax: (256) 722-0985
E-mail: ron.oguin@oarcorp.com

Malcolm Morrison, Ph.D., is a senior software engineer for OAR Corporation with 15 years experience as a developer of information and weapon systems, an educator, and consultant. His focus has been on software process management impacts. Morrison has served as a full-time faculty member at the University of Alabama in Huntsville and Salisbury University in Maryland. He is a developer of the Fire Control-Node Engagement Technology Technical Fire Control Architecture.

OAR Corporation
4910-L Corporate Drive
Huntsville, AL 35805
Phone: (256) 842-6937
Fax: (256) 722-0985
E-mail: malcolm.morrison@oarcorp.com

Deborah A. Butler is an electronics engineer with the Aviation and Missile Research Development and Engineering Center with more than 15 years experience in the design, development, and fielding of real-time military embedded applications. As an experienced hardware and software designer, Butler has contributed to the successful design, development, and demonstration of programs such as the Future Missile Technology Integration Program, Long Range Fiber Optic Guided missile, Future Artillery Loiter Concept, Low Cost Precision Kill, and Compact Kinetic Energy Missile. She has an extensive background in missile system research and development and is the program manager for the Fire Control-Node Engagement Technology Science and Technology Program.

U.S. Army Aviation and Missile Command
ATTN: AMSAM-RD-MG-NC
(Deborah A. Butler)
Redstone Arsenal, AL 35898-5000
Phone: (256) 876-1303
Fax: (256) 842-9476
E-mail: deborah.butler@rdec.redstone.army.mil

WEB SITES

Office of Naval Research Science and Technology

www.onr.navy.mil/sci_tech/jtf_warnet/public/default.asp

The Joint Task Force WARNET leverages the successes of the Extending the Littoral Battlespace Advanced Concept Technology Demonstration, which included a highly mobile wireless wide-area relay network in support of tactical forces, designed to revolutionize joint expeditionary warfare. It demonstrated a precise, near real-time, common tactical picture to squad level, facilitated dynamic maneuver, enhanced naval fires, and reduced fratricide.

Automation and Robotics Research Institute's Enterprise Engineering

http://arri.uta.edu/eif

The Enterprise Engineering Program is part of the Automation and Robotics Research Institute at the University of Texas, Fort Worth. The Enterprise Engineering Program's mission is to research, develop, and deploy methods, philosophies, and tools for the implementation of the integrated enterprise. Its objective is to develop the enterprise engineering discipline and to develop reconfigurable manufacturing concepts for the agile enterprise.

Command and Control Research Program

www.dodccrp.org

The Command and Control Research Program (CCRP) within the Office of the Assistant Secretary of Defense focuses on (1) improving both the state of the art and the state of the practice of command and control and (2) enhancing Department of Defense understanding of the national security implications of the information age. The CCRP pursues research and analysis in command and control (C2) theory, doctrine, applications, systems, the implications of emerging technology, C2 experimentation and develops new concepts for C2.

The American Society for the Advancement of Project Management

www.asapm.org

The American Society for the Advancement of Project Management (ASAPM) is a not-for-profit professional society dedicated to advancing the project management discipline. Working with members, ASAPM provides the leadership for professional growth of both members and the profession. ASAPM is working at the forefront of the project management discipline to push practices, procedures, and techniques to their best use. ASAPM will soon be offering a project manager (PM) certification, and also expects to soon sponsor a federal government-oriented PM conference.

Global Combat Support System

www.disa.mil/ca/buyguide/prodsrvcs/gcss.html

The Global Combat Support System Combatant Commanders/Joint Task Force was developed by the Defense Information Systems Agency to respond to the operational concept of focused logistics articulated in Joint Vision 2010, and reinforced in Joint Vision 2020. Focused logistics is the fusion of logistics information and transportation technologies for rapid crisis response; deployment and sustainment; the ability to track and shift units, equipment, and supplies; and the delivery of tailored logistical packages directly to the warfighter.

High Availability Linux Project

www.linux-ha.org

The High Availability (HA) Linux project goal is to provide a high availability solution for Linux that promotes reliability, availability, and serviceability through a community development effort. This site is a collection of information for the Linux-HA project, including a how-to on Linux-HA, available downloads, communication architectures, commercial software, and more.

CALL FOR ARTICLES

If your experience or research has produced information that could be useful to others, CROSSTALK can get the word out. We are especially looking for articles on **Best Practices** and **Lessons Learned** for upcoming issues. Below is the submittal schedule for three upcoming issues:

Software Consultants and Mentors
February 2004
Submission Deadline: September 15, 2003

System Assessments
March 2004
Submission Deadline: October 14, 2003

The Software Edge, Enabling the Warfighter
April 2004
Submission Deadline: November 17, 2003

Please follow the Author Guidelines for CROSSTALK, available on the Internet at: <www.stsc.hill.af.mil/crosstalk>. We accept article submissions on all software-related topics at any time, along with Open Forum articles, Letters to the Editor, and BackTalk submissions.

Enterprise Engineering: U.S. Air Force Combat Support Integration

Eric Z. Maass
Lockheed Martin Mission Systems

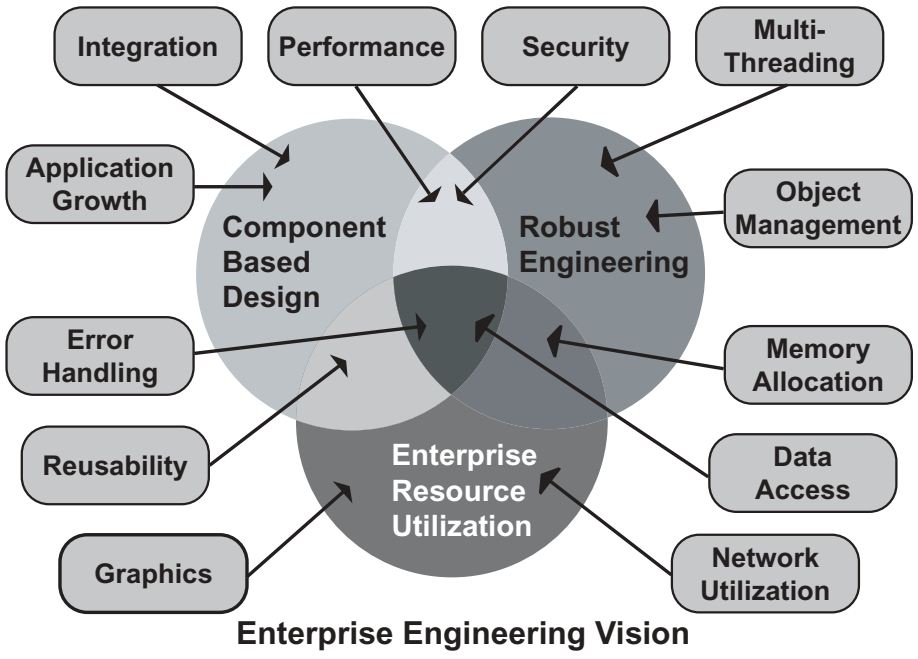
Enterprise engineering is quickly becoming a hot term used to describe the future movement of software engineering. This movement, for both management and development organizations, can be daunting. Application developers will find though that this model is useful in building a common enterprise from disparate organizations. This article covers fundamental considerations for developing to an enterprise engineering vision and discusses basic techniques of enterprise application development as lessons learned on the U.S. Air Force's Global Combat Support System enterprise.

The term enterprise engineering describes a large gamut of engineering practices and processes that enable an organization to design, develop, stand up, and maintain an enterprise-computing environment. Facets of enterprise engineering might range anywhere from models such as time-based competition, continuous improvement, and business process reengineering to the real *meat and potatoes* of enterprise application design, deployment, and integration (enterprise application integration) [1]. However, from a technical engineering perspective, how does a large organization tackle the daunting task of integrating numerous enterprise-class applications while maintaining a single engineering vision? Furthermore, how do multiple sub-organizational structures work to develop first-class enterprise applications for a single, common infrastructure, enterprise environment?

The first realization that must be addressed is that enterprise engineering is the future of the software engineering world. Software, whether it is designed for a home computer, large computing environment, or mobile computing device, is becoming increasingly powerful due to its ability to interact, or integrate, with other systems and their respective software. For instance, after placing an order to purchase an item on an Internet-based store, typically the buyer would receive an e-mail with the status of the order. To make this happen, it could mean integration of several systems and their software: a credit card processing system to verify the credit card used, a warehouse system to see if the item is in stock, a third-party delivery system to prepare delivery of the item, etc.

In the past, these systems were either not integrated or were integrated in some proprietary manner, which made them very costly and not very flexible. Today, the idea of enterprise engineering is sweeping the software engineering world with technologies such as Java 2 Enterprise Edition (J2EE), Web services, and enterprise application integration tools. Hence, we have the introduction of enterprise engineering – the vision of engineering software and systems that integrate across various systems, use open standards, and bridge the gaps between legacy, stovepipe system mentalities (see Figure 1).

Figure 1: *Enterprise Engineering Vision*



In the case of the Air Force, building an integrated combat support enterprise is key to providing real-time, accurate, integrated information to the warfighter. Allowing stand-alone combat support systems to integrate in an efficient, robust, secure manner vastly improves the value of information available to those making combat decisions. For example, being able to integrate combat support systems could provide a single user with the capability of seeing not only what planes are available for a mission, but also their mission readiness, available personnel, financial data relating to making that mission happen, and so forth. Previously, this data might only be available through many separate systems that might have conflicting information due to the fact that they were not integrated and shared the same real-time information.

Now that the end goal is clear, the question becomes how to get there. A key to success in such an enterprise engineering environment is to become a heavily integrated organization built upon common services and open standards, and employing rigorous application development standards in the software engineering processes of the applications joining the enterprise. Of course, there are multitudes of accompanying business models that will wrapper and compliment these engineering practices.

The U.S. Air Force is currently creating such an enterprise by building an integrated environment – one enterprise – that is the platform of integration for the Air Force's vast combat computing systems. The enterprise, known as Global Combat Support System-Air Force (GCSS-AF), draws upon currently operational mission

applications in the combat support arena to modernize applications in accordance with a new integrated enterprise engineering vision. In doing so, from a pure engineering perspective, there are many obstacles to contend with – especially when coordinating the efforts of multiple software development organizations to integrate under one enterprise.

One of the first technical obstacles is enabling developers to understand the vision! Within that vision are the technical details regarding the enterprise architecture, its shared and common services, components and methods of integration, compliance standards, and much more. This article looks more closely at a subset of these aforementioned aspects of an enterprise, some commonly trusted best practices when developing to such an enterprise, and how these issues relate directly to the Air Force's GCSS-AF initiative. The enterprise engineering vision, being the super-set of the technical aspects that will be discussed within this article, can be viewed in Figure 1.

Building an Enterprise Engineering Vision

Building an enterprise engineering vision is one of the most important milestones of successfully constructing an integrated enterprise environment on such a grand scale as the U.S. Air Force combat support computing structure. The enterprise engineering vision is fundamental to ensuring that disparate software development organizations understand their role in the enterprise.

The GCSS-AF program delivers the technical components of its enterprise engineering vision through a platform – an infrastructure for development – known as the Integration Framework (IF). The IF, a conglomerate of commercial off-the-shelf products in a n-tier, Web-based, J2EE-enabled architecture, provides a set of common services and components for applications that join the enterprise. The framework provides a *living space* for application integration that enforces standards while providing a way for applications to join the enterprise and reduce the cost of software development by avoiding reintroduction of common services (such as security, messaging, and data warehousing).

The diagram depicted in Figure 2 demonstrates a typical set of common services offered by the IF on GCSS-AF in a four-tier enterprise. These tiers represent a set of common services that are available to application developers as part of their design for joining the enterprise.

Figure 2: *The Integration Framework on GCSS-AF*

The GCSS-AF environment, designed to host a wide variety of disparate applications in a manner that is conducive to forming a single, heavily integrated enterprise, is also faced with an interesting engineering case: How does an organization collectively ensure that such a conglomerated environment operates efficiently and successfully?

Modern Problems: A Return to Basics

The idea of enterprise engineering may be new to many development organizations, but the fundamentals of making it *work* are largely based on a model of software engineering practices that far outdate the modern concepts of enterprise engineering and most of the technologies that may be present in that environment.

Each application is part of a phased approach at reaching a vision of an integrated enterprise. This means that enforcing the use of the technologies present in the enterprise will be key to following the enterprise engineering vision; on that same note, flexibility in the technology set present in the enterprise is also important. However, perhaps even more important is ensuring that these services and technologies are being implemented correctly. This brings us back to the basics!

Java may be a relatively new programming language, but it shares much in common with its ancestors. This is relatively true for most modern technologies – they display tendencies and traits from their ancestors that are important to note because they may largely aid in successfully implementing them in similar or new ways than was done previously.

Let us quickly review some of the basics of good software engineering practices and see how these might be applied to modern-day enterprise engineering in an environment such as GCSS-AF.

Component-Based Design

Every modern developer has heard the term component-based design. The real question is how many modern developers fully extend its theory into their practices? Furthermore, how many modern enterprise developers and architects consider component-based design theories an integral part of their work? The answer is always quite simple – not enough! [3]

When we speak in terms of Java enterprise development, component-based design should be one of our first thoughts. Java, being a multiplatform-compatible programming language, employs technologies that tend to slow code execution (in comparison to older, single-platform languages) which, at the same time, makes the language flexible. Unlike languages in the past, Java therefore requires additional attention to component-based design so that the applications created in these environments can perform on par with older, quicker languages.

For example, take the following pseudo code design of procedures in a Pascal-like application (see Figure 3, page 18) that might be designed as follows:

```
Function calcTotal (var userTotal : integer);

Begin
        grandTotal := userTotal + (userTotal * salesTaxPer);
        writeln 'Your total is: ' + grandTotal;
end; {calcTotal}

Function on MailingList (var mail : boolean);

Begin
        {... lookup if customer is on mailing list ...}
        {... access private account data ...}
end; {onMailingList}

Procedure initProgram;

Begin
        writeln 'XYZ Company System';
        if user.onmailinglist = true then
                initShoppingChart;
end;
```

Figure 3: *Pseudo Design Example: Pascal-Like Application*

This pseudo code snippet demonstrates a function and procedure call in a Pascal-like application. The function, *CalcTotal*, is used to calculate a user's shopping total by adding the necessary sales tax and perhaps other related calculations. It then displays the grand total.

The second, a function called *onMailingList*, is used to determine whether a customer is on the company's mailing list. The procedure does some set of operations (most likely accessing some kind of database) and would return a true or false Boolean value reflecting the customer's mailing list status. Next, we see that the function accesses some private account data related to the customer previously looked up for membership on the mailing list.

Finally, we have a procedure that initializes the application. It most likely would perform multiple operations to initialize the application such as displaying the company name. In addition, this procedure also determines whether or not to initialize the user's *shopping cart*.

Agreeably enough, the above example is not a great example of component-based design, but in the case of a stand-alone, single-user Pascal-like application, the above would probably not be a terrible setback for application performance. In an enterprise engineering environment like GCSS-AF, however, this example would be a major contributing factor to deterioration of system resources, application performance, and overall enterprise engineering vision for high-performance first-class integration because the key fundamentals of sound, component-based design have been overlooked.

In an enterprise with a vision of full application integration, certain items are of key concern [4]. These are explained in the following sections.

Performance

Component-based design is *necessary* for obtaining higher levels of enterprise application performance. Separating out code that is unnecessary for execution will help mitigate the problems seen in slower languages such as Java.

As seen in the example, had the programmer separated out the arbitrarily placed initialization of the user's shopping cart and moved this functionality to a more appropriate part of the code, the application would initialize quicker. For instance, the programmer might have decided to determine whether to initialize the shopping cart only after the user elected to *go shopping* rather than just assuming the user was ready to shop.

> *"Today, the idea of enterprise engineering is sweeping the software engineering world with technologies such as Java 2 Enterprise Edition (J2EE), Web services, and enterprise application integration tools. Hence, we have the introduction of enterprise engineering."*

Reusability

Component-based design is also *necessary* for obtaining code reusability. Although this may not be particularly necessary in all initial instances, having code in a reusable format allows for quick transitioning to higher levels of integration such as allowing an application to offer parts of its functionality as a Web service.

In the earlier example, the programmer should have broken out the function CalcTotal into two separate functions: one for calculating the total, and another for displaying the results. This micro-managed modular approach would (on a grander scale) help the programmer quickly share functionality and reusability of that functionality in the future.

In the world of enterprise engineering, this becomes especially important and is an intricate part of the J2EE model. The earlier example would therefore translate to having all Hyper-Text Markup Language generated in Java Server Page code for displaying the total while the calculation itself would take place, perhaps, in a servlet or Enterprise Java Bean.

Security

Security is often overlooked and becomes an afterthought in application design. Component-based design is also necessary for ensuring that the proper security is in place for an enterprise application.

In the previous example, we see no indications of security measurements. However, we might assume that in an enterprise application (especially one residing in the GCSS-AF environment), sensitive data will need to be filtered, restricted, and monitored. Therefore, implementing method-level security is, typically, necessary in all applications.

In the function onMailingList, private customer data is accessed after the user has been looked up on the mailing list. If this functionality is not necessary for determining whether a user is on the mailing list or not, it should be broken out into a separate, secured procedure. Even if the private customer data is not made available to the application user in the procedure onMailingList, an application error, for example, could cause unexpected exposure of the data or privileged functionality.

Learning to Live Inside the Box

Generally speaking, we would like application developers to *think outside the box* while still realistically considering that their application must *live inside the box*. Enterprise engineering is ultimately the balance between the two.

When applications decide to join the GCSS-AF enterprise, multiple considerations need to be made in order to account for the application's capacity resources, performance requirements, compatibility with other GCSS-AF applications, and a multitude of other facets that may impact the application's ability to reside onboard the program. While a good number of these considerations may be business-process related, another good number is purely engineering issues that must be addressed during an application's design phase.

Again, we are taking some steps back

to the basics, but emphasis on these techniques and viewpoints may, in the end, determine an application's success on GCSS-AF or any other enterprise environment [4].

Memory Allocation

Allocating too little or too much memory is often not detrimental to a smaller standalone application; however, when in a J2EE environment, memory handling becomes increasingly important as both the application and enterprise weigh in [4].

Common memory allocation problems are as simple as using efficiency when dealing with data types in an application. For instance, allocating a 30-character array for a 10-digit code may waste 40 bytes of memory per user executing that code snippet. With potentially thousands of users executing that same code simultaneously, one simple programming error due to negligence could lead to a significant amount of memory waste. An application containing many of these same mistakes in conjunction with other forms of memory allocation errors could easily bring down the application and other applications in the enterprise that are either dependent upon shared resources or services provided by this application.

Multi-Threading

Multi-threading is an application architecture design point intended primarily to allow an application to perform multiple tasks at once in a safe, highly efficient manner.

Multi-threading an enterprise application is considerably important. Most J2EE operations performed in a Web-based application environment should be designed as asynchronous, multi-threaded calls. Depending on synchronous operations can drastically impede an enterprise's performance [2].

Data Access

Remember that your *common services* are generally shared resources that are accessed by various other applications. This includes your data resources. It is important to keep in mind that database connections should be pooled, take advantage of extensible architecture (XA)-compliant database drivers, and be used as efficiently as possible.

A few examples of this may include the following: accessing a database connection, Transmission Control Protocol/Internet Protocol connection, File Transfer Protocol connection, or other connection type to an external resource only when necessary [2]. As well, when connected to the data resource, make sure data are created, read, updated, and deleted in the most efficient natures. For instance, search a table based upon an index – avoid scanning the entire table. Other considerations may include terminating connections when not in use, simplifying data storage schemes (size and complexity of records), transferring only necessary parts of a record, and using the smallest, most efficient data types in records (i.e., the abbreviation AL instead of Alabama).

Error Handling

A single application's stability can potentially impact the stability of the enterprise. For example, if an application's faulty code continually tries to poll an enterprise resource every second with a large query due to a failure in the application, the result would be troublesome for the enterprise resource and every other application depending upon the availability of that resource [4]. Many such problems can be avoided with the use of rigorous error handling and capturing [3]. Employing tight regulations for error handling when performing operations that may impact the enterprise is extremely important and must be designed into the application's functionality.

"Building an enterprise engineering vision is one of the most important milestones of successfully constructing an integrated enterprise environment on such a grand scale as the U.S. Air Force combat support computing structure."

Clean Up

Garbage collection is Java's native way of conserving resources [2]. However, performing such maintenance as garbage collection is the obligation of design. Unused objects should be discarded to conserve memory resources; database records that are no longer necessary should be deleted; databases that are frequently changed should be *reorganized* for performance. Keeping an application's workspace and footprint small, tight, efficient, and clean will help the entire enterprise be successful!

Help Keep the Roads Clear

Network congestion is one of the top causes of poor Quality of Service (QoS). In an enterprise, QoS is a shared responsibility that starts with the QoS initiatives of each application residing in the environment.

A first-class enterprise application is just as concerned with QoS as it is with functionality and robustness of code. QoS typically includes everything from usability of user interfaces to responsiveness of requests to the application; however, one major aspect of QoS that will hit every application hard will be network congestion.

Typically, as an enterprise application residing in a conglomerated environment, the standards or availability of network resources can be somewhat questionable. Therefore, cleverly designing your application to avoid potential QoS problems is typically a wise decision. The following sections are a few examples of design aspects that would be beneficial to an enterprise application's QoS.

Graphics

Graphics are notorious for causing poor QoS and are generally unnecessary for most applications. If your application must employ graphics, some general guidelines should be followed:

- Use black and white graphics if possible.
- If color graphics are necessary, use the lowest bit-depth possible (8-bit, 256 colors) to reduce image size.
- Use the highest compression available on file formats. Using JPG or GIF formats above BMP formats is such an example.
- Keep the image dimensions as small as possible while still keeping effective its business purpose.
- Allow users the option of viewing graphics instead of displaying them by default.
- Display a minimum number of graphics per page.

Packet Trips

Avoid using multiple round trips to achieve what could be done in a more efficient, perhaps larger, transmission. This also includes reducing the amount of data that is retransmitted or checked for

integrity.

A simple example in a Web-based application would be static data retransmission. Employing frames in an application can maintain static data on the user's screen while allowing only the necessary frame to update with a request.

Data Transmission

Send and collect only necessary data. Avoid sending or collecting extraneous data that would utilize enterprise resources without gainful purpose. Some examples of this may include the following:

- Large cookies containing consequential or infrequently used data.
- Dynamic Web objects that require a high frequency of refreshing (i.e., syndicated news).
- Related but unrequested data. Giving the user options to view this data conserves resources rather than sending this data by default.

Keys to Success

From a developer's standpoint, enterprise engineering in a conglomerated enterprise is not an easy task. However, as we have reviewed here, sometimes taking a step backwards and understanding the basics is the key to providing a strong foundation for an enterprise application. There will inevitably be a gamut of hurdles to jump over regarding application-specific engineering, business process models, and so forth; but, attacking the basic engineering from these guidelines and building these guidelines into the application's model will also inevitably aid the success of the application and the enterprise as a whole.

In review, the points to remember include the following:

- Design to the enterprise engineering vision. Using (correctly) the resources available from the enterprise will avoid unnecessary development efforts and help an application integrate into the rest of the environment.
- Practice healthy, component-based design techniques. Component-based design has extremely important benefits in the enterprise, including performance, code reusability, and security.
- Pay rigorous attention to detail. Rigorous attention to programming details will affect the success of an enterprise application to a much greater degree over a stand-alone application.
- Conserve network resources. Enterprise applications must share network resources. Paying attention to these details during design time will allow all applications in the enterprise to increase their QoS.

Enterprise engineering is the future of large-scale organizational computing. Understanding how to develop well performing, integrated applications for such an environment is the beginning step for a successful experience.◆

References

1. Enterprise Engineering: An Information Systems Perspective. 27 Feb. 2003 <www.eil.utoronto.ca/ papers/ mikePapers/eeg16.html>.
2. Farley, Jim, William Crawford, and David Flanagan. Java Enterprise in a Nutshell. Sebastopol, CA: O'Reilly and Associates, 2002.
3. Joines, Stacy, Ruth Willenborg, and Ken Hygh. Performance Analysis for Java Web Sites. Boston, MA: Pearson Education, Inc., 2003.
4. Maass, Eric. "Application Performance for GCSS-AF." GCSS-AF Guide to Developing With the Integration Framework. June 2002.

About the Author

Eric Z. Maass is a software systems engineer for Lockheed Martin Mission Systems in Owego, N.Y., a Capability Maturity Model Level 5 organization. As a member of the software integration and development team on the Air Force's Global Combat Support System (GCSS-AF) program, Maass' primary responsibilities include leading and supporting architecture, development, and integration of the enterprise's security services, leading systems performance optimization research for the GCSS-AF production enterprise, and providing engineering support for application integration into the GCSS-AF program. Maass was recognized in 2002 as a GCSS-AF Top Contributor. He is a graduate of Syracuse University.

Lockheed Martin Mission Systems
1801 State Route 17C
MD 0605
Owego, NY 13827
Phone: (607) 751-2293
Fax: (607) 751-2538
E-mail: eric.maass@lmco.com

Technical Reference Model for Network-Centric Operations

Bradley C. Logan
The Boeing Company

The majority of today's weapon systems are platform-centric; they work well within the same weapon system's environment, but do not readily collaborate with other weapon systems. The Strategic Architecture Reference Model (SARM) is a communication and information architecture framework based upon commercial and government interface standards. Organized to address system-wide network design issues, such as information assurance, the SARM is an enabling technology framework to allow platforms and systems to interface to the Global Information Grid as interoperable nodes on the network. This article discusses the benefits of having a SARM for platforms and systems, what is done with it, what should be in it, how to understand its structure, and how to use the SARM.

One thing agreed on among military strategists is that dominance on the 21st century battlefield will be driven by information superiority. Those who generate, manipulate, and use information in a precise and timely manner will dominate the battlefield of the future. The key to such superiority is network-centric warfare:

> Network-Centric Warfare (NCW): We define NCW as an information superiority-enabled concept of operations that generates increased combat power by networking sensors, decision makers, and shooters to achieve shared awareness, increased speed of command, higher tempo of operations, greater lethality, increased survivability, and a degree of self-synchronization. In essence, NCW translates information superiority into combat power by effectively linking knowledgeable entities in the battlespace. [1]

However, NCW is not just about connecting weapon systems together on a communications network. It is about utilizing the connectivity of the network to transform operations doctrine. This is done by rapidly gathering raw data from across the network, then fusing it together to transform data into information about the battlespace. This correlation of information from across the network transforms it into an understanding of the battlespace threats and assets. NCW is about the timely utilization of that knowledge of the battlespace state and events to rapidly make better-informed decisions, both proactive and reactive. NCW is about letting computers do what they do best, moving and manipulating data, and letting humans do what they do best, making informed decisions.

For example, recent operations in Afghanistan and Iraq touched the tip of the iceberg for transformation with voice communications letting a soldier on the ground request and guide air strikes from air assets flying combat air patrol missions. Network-centric operations (NCO) is about speeding up that process through automation with the players as networked nodes; with intelligent software taking sensor data in, analyzing it, looking for the best effector assets on the network; and dispatching tasking orders in a fraction of the time.

> *"Network-centric warfare is about letting computers do what they do best, moving and manipulating data, and letting humans do what they do best, making informed decisions."*

While this article focuses on military applications, the principles of NCW apply to civilian applications such as police and fire-first responders. For the broader application, we use the term NCO.

Stovepipe Systems

While some weapon systems can work together, most of today's deployed systems are islands of self-contained connectivity, or *stovepipe* systems. That is, those weapon systems components that were designed at the same time to work together can communicate and exchange data, but that is the extent of their network connectivity. At best, communication with a disparate weapon system developed at a different time on a different contract is difficult and time consuming. This is a vastly different vision compared to NCW where weapon systems rapidly and easily work together in large Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance systems.

The proliferation of wireless communication systems using different protocols, the difficulty for coalition forces to communicate over such systems, and the difficulty of coordinating both police and fire activities on Sept. 11 are examples of the present state of stovepipe systems both military and civilian. The enabler for NCW is the *interoperability* of disparate weapon systems to form systems of systems (SoS).

> Interoperability: The ability of systems, units, or forces to provide services to and accept services from other systems, units, or forces, and to use the services so exchanged to enable them to operate effectively together. [2]

Boeing Strategic Architecture Initiative

Moving to NCW via significantly increased levels of interoperability will be a transformational process. The Boeing Strategic Architecture organization was created and chartered to integrate all of Boeing's platforms, systems, and programs into a single common communication and information framework.

The main thrust of the organization is to create, control, and disseminate the Strategic Architecture Reference Model (SARM), a communication, information, application, and presentation architecture framework. An enterprise-wide central organization that has access to all programs and a cross-program perspective

ensures a system-wide architecture to directly address key network and node design issues.

The Strategic Architecture organization is also forming an industry consortium[1] of infrastructure providers and users to promote adoption of the framework across non-Boeing products and to ensure the framework is developed and evolves with the best industry practices and products. The intention is to create open industry standards of the interoperability infrastructure lower levels via the consortium. Contractors then compete at the higher levels of the model where their application domain expertise provides added value and the open infrastructure provides a common foundation upon which the applications are built. Thus, the SARM is an enabler for SoS interoperability.

Vision to Achieve Information Superiority

The Global Information Grid

Computer networks are transforming business processes globally by allowing closer and more rapid collaboration and coordination both internally and externally among a business, its suppliers, and its customers. Timely network access to data from its business environment allows executives to correlate, fuse, and transform the data into critical operating knowledge used to make timely informed decisions, and allows using the same network to disseminate directives to effect change to achieve business goals.

This paradigm applies across organizations where data are gathered, processed, and acted upon: commercial businesses, civil service, and the military. While networks such as the Internet may be suitable for many applications, the military has unique and stringent needs in its business environment.

The Global Information Grid (GIG) is the vision of the assistant secretary of defense for Command, Control, Communications, and Intelligence for achieving information superiority. The GIG is a single, secure grid providing seamless end-to-end capabilities to all warfighters, national security, and support users. It supports the Department of Defense and intelligence community requirements from peacetime business support through all levels of conflict. The GIG provides plug-and-play interoperability for the joint services and coalition users with high capacity network operations. It also provides interoperability at the strategic, operational, tactical, and base/post/camp/station levels [3, 4].

Operational Benefits of NCO

The power of information superiority achieved by networking assets together can be illustrated by analogy with phased array technology. A single non-directional sensor by itself may detect the presence of an object, but putting two sensors together with time-of-arrival measurement capability allows the raw data of the two sensors to be correlated to yield a direction for the object. As more and more sensors are added, the precision of the location data increases. The data have been changed into more robust information about the object, which enables refined object tracking.

In NCO, capabilities for sensing, commanding, controlling, and engaging are robustly networked via digital data links. The source of the increased power in a network-centric operation is derived in part from the increased content, quality, and timeliness of information flowing between the nodes in the network. This increased information flow is key to enabling shared battlespace awareness, and increasing the accuracy of the information. These operational benefits are derived from having the GIG enabling technology represented by the SARM guiding the development of nodes that plug and play on the network; they are not benefits of the SARM itself.

Reference [1] provides a much more complete discussion on the benefits of NCO.

Technical Approach

Why Have A SARM?

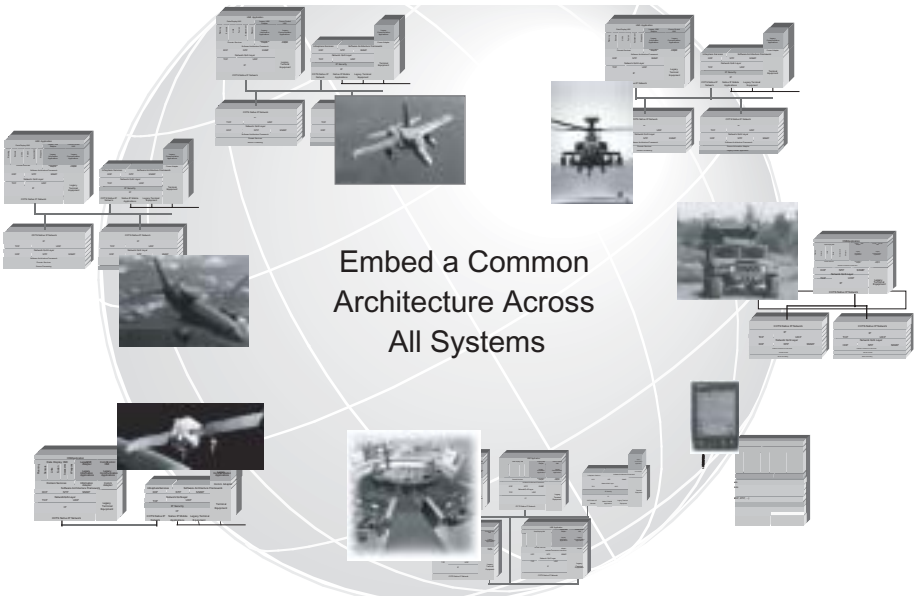
Network System Design: To work well, the fundamental architecture of networks and their nodes are designed together as a system. Creating and managing the SARM can ensure the following:

- The components used to build up the SARM are integrated into a system-wide architecture so that fundamental network system design issues such as information assurance are addressed by the design as a whole from the beginning.
- The single consistent framework of the SARM is implemented on all platforms and systems connecting to the network.
- This system-wide architecture will enhance the ease of integration of platforms and programs as nodes on the network and the level of interoperability between them, while maintaining the precepts of information dissemination control within an information assurance doctrine.

Reusable Components: The universal use of the products in SARM instantiations will foster the creation of reusable components that provide common data and functionality across platforms and systems. This will bring the expected benefits of decreased development costs, faster time to market, extensive use of commercial off-the-shelf (COTS) products, decreased maintenance costs, open standards, and robust products suitable for many environments.

The goal is to quickly get to the point where the SARM guides node interface design based upon a product catalog of qualified and tested products that form instantiations of the framework. This will enable many programs to take the products and use them directly off the shelf.

Figure 1: *Notional Deployment of SARM With Different Implementations Suited to Each Platform*



This will free the programs from spending resources on what should be common infrastructure, and instead allow them to concentrate their assets on solving their unique programmatic challenges.

What Will You Do With a SARM?

Figure 1 provides a notional idea of what you do with a SARM: instantiate the framework across the platforms and systems that are to become nodes on the network. The photos represent the platforms and systems, while the boxes to the sides represent different instantiations of the SARM – some larger, some smaller. Realize that each of the platforms may have different needs for interoperability and therefore different instantiations of the SARM. For example, the kind and amount of information needed by a hand held device will differ from systems needed by an operations center.

What Do You Want in a SARM?

Based Upon Standards: A fundamental design decision in creating the SARM is to base it upon standards such as the Internet Protocol (IP) [5] as the basis for the infrastructure. With the rapid and far-reaching success of the Internet, this brings many benefits, including open commercial standards, multiple competing sources for compatible products resulting in reduced costs and increased maintainability, mechanisms for technology insertion, and wealth of existing application technology guidelines for robust product development. The SARM also promotes the use of government standards, including the Joint Technical Architecture [6] and the Defense Information Infrastructure Common Operating Environment [7].

Common Interface and Functionality: To be part of a network, a platform must comply with the common interfacing standards defined by the network. The Internet works because every node on the network complies with the agreed-upon standards for basic connection and data exchange. This infrastructure can be common among the nodes in the network and be the foundation upon which applications residing on the network nodes are based.

Common Ontology: Once the physical connection is made to the network and the data moves through the communications layers of a node to become IP packets delivered by the operating systems to applications, the remaining key to interoperability is the consistent syntax and semantics of data that are exchanged on the network. This is referred to as *ontology*, an explicit formal specification of how to represent the objects, concepts, and other entities that are assumed to exist in some area of interest along with the relationships that hold among them. The richness and extent of the ontology supported on a platform relates to the level of interoperability on the network supported by that node.

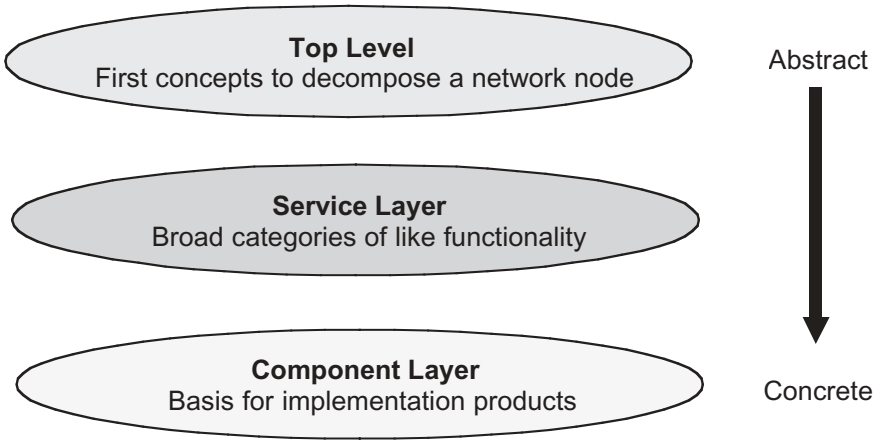


Figure 2: *Discussion of SARM Structure Follows a Top-Down Design Paradigm*

Understanding the SARM

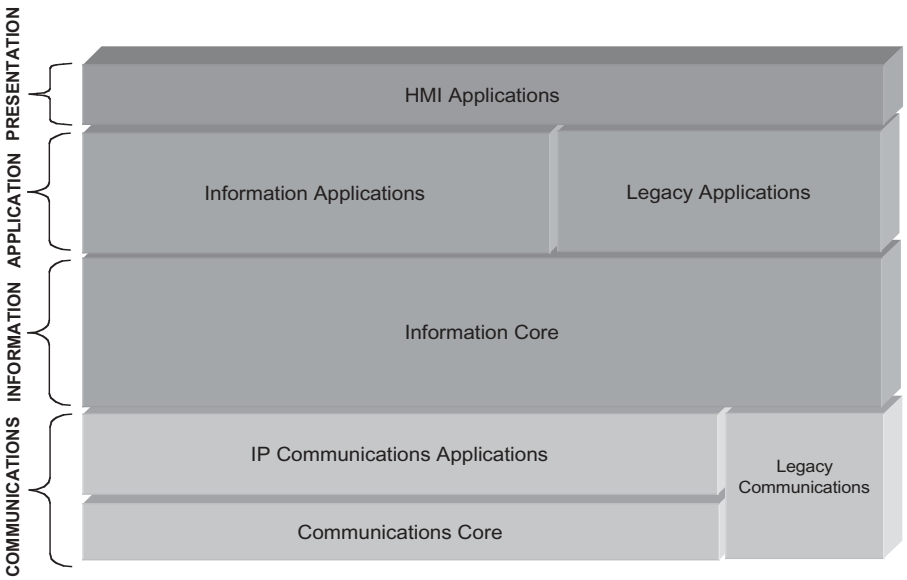
Hierarchy Diagrams: The discussion of the SARM follows a top-down approach, as shown in Figure 2, from an abstract decomposition of the functional units of a network node to specifications for the component pieces used to implement the functionality.

Top Level: The SARM is a hierarchical structure with increasing levels of detail and specificity at each successive level. The first level serves to divide the network node into broad categories of functionality and responsibility as shown in Figure 3 and as follows:

- The *communications* layer represents essential communications functions and services provided by the IP-centric network. Provision is made for legacy communications mechanisms to allow them to become nodes on the network.
- The *information* layer provides services that support the interchange and management of information between applications and the external environment. The key to this layer is a common ontology for the information flowing on the network and residing on the nodes.
- The *application* layer implements program-specific functional processing, e.g., position/navigation, sensor, control of real-time systems, and analysis of order of battle.
- The *presentation* layer implements program-specific human-machine interface (HMI) requirements.

The Strategic Architecture organization concentrates its efforts on the com-

Figure 3: *Top Level of the SARM Divides the Model Into the Major Functional Areas*



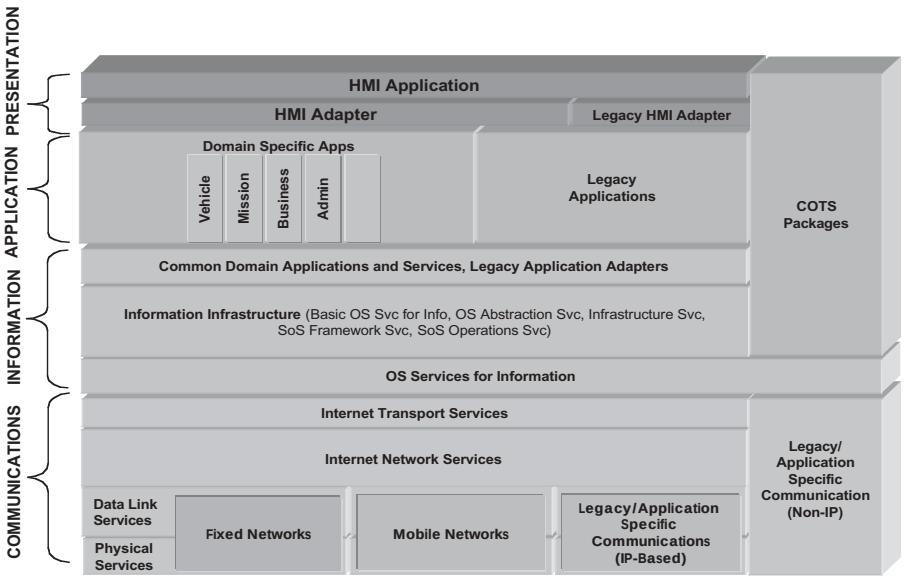


Figure 4: *Service Layers Further Decompose the Framework Into More Specific Functional Categories*

munication and information layers since these are the common infrastructure layers that enable interoperability between nodes. The application and presentation layers are more specific to the needs of a program. However, when application functionality that is common across programs is identified and data models and methods manipulating that data are generalized, these data and functions can be moved into the information layer for reuse across programs.

Service Layer: The next level further refines the decomposition into major services within the top-level decomposition and is shown in Figure 4.

- The *communications* services follow the layered guidelines of the Open System Interconnection (OSI) seven-layer model and the IP model, but implement only the bottom layers: physical, data link, network, and transport services. The diagram shows further decomposition into a variety of networks dependent upon the mobility of the platforms. Legacy communication systems are supported, both those that are IP and non-IP based.
- The *information* services are not strictly layered as the communication layers. The information services are application program interfaces (API) that provide operating system services, data management, information assurance, and similar services that perform fundamental control, access, and manipulation of information in a networked SoS model.
- The *application* and *presentation* services are notional in this diagram since the efforts of the Strategic Architecture organization concentrate on the other layers and leave these layers to the programs for their value added. The intent is that program-specific processing and functionality is implemented in a network node at these levels.

Component Layer: At present, the SARM diagrams go down one more level in the communication and information layers to decompose them into components. The idea is that each of the components would be implemented or mapped to COTS or government off-the-shelf (GOTS) products that provide the functionality defined by the component. The components are intended to be terminal or leaf nodes in the hierarchy tree. The software components of the SARM will have standard APIs defined for them so that applications may call the services independent of the implementation details. The common defined API will make the components independent of the underlying implementation and the hardware/software platform on which it executes. In this way, applications may be written to depend upon a platform-independent environment provided by the SARM.

"While some weapon systems can work together, most of today's deployed systems are islands of self-contained connectivity, or stovepipe systems."

At this level of detail, the layered communications services model progresses in the protocol stack (read from bottom up) from the physical, data link, network services, and transport services. Provision is made for fixed location, mobile, and legacy systems as network nodes. Network quality of service and information assurance components exist in the layers as part of the overall design.

The structure of the information services layer progresses from basic operating system, to SoS services that provide underlying infrastructure for information management in a distributed networked environment. For example, a component provides networked directory services such as the Lightweight Directory Access Protocol.

Using The SARM

Component Catalog and Portal: Products that implement the functionality of the SARM components are being collected from COTS, GOTS, and other sources and put into a database along with tested configurations of the products that can be used to construct instantiations of the framework on different platforms. A Web portal interface is being developed to interface with the database. The portal interface will help users search for combinations of products that meet a program's functional needs to become a node on the GIG.

SARM Evolution

The SARM is not complete and it will never be complete. It represents an expandable framework that will evolve with technology and time. Each of the layers in the reference model will expand at least horizontally to include new technologies fulfilling the same kind of functionality as existing services and components, while vertical expansion would include possibly new common functional capabilities. For example, as new communication protocols are developed with higher bandwidths, lower latency, and higher levels of information assurance, these can be added to the communication layer in the fixed or mobile networks.

Populating the SARM

One organization should be the custodian of the SARM, but it alone does not create the SARM and does not populate the component catalog solely on its own. This is an industry-wide effort that spans programs that act to supply products to implement components as well as use

those supplied by others. The NCO consortium is a critical part of proliferating this framework across the industry and will be the long-term custodian of the standards.

Conclusions

The information age is transforming business practices with the ability to network organizations internally and externally to their customers and suppliers. The military sees the need to follow similar paradigm shifts with the vision of a GIG where information flows securely between sensors, effectors, and decision-makers in the battlespace for unparalleled degrees of collaboration. To enable this level of interoperability requires a network system-wide guiding framework and products that implement that framework. The Strategic Architecture Reference Model addresses those issues and will enable NCW by providing the infrastructure for platforms and systems to become nodes on the GIG.◆

References

1. Alberts, David S., John J. Garstka, and Frederick P. Stein. Network Centric Warfare: Developing and Leveraging Information Superiority. 2nd ed. (Rev.) <www.c3i.osd.mil/NCW/ncw_0801.pdf>.
2. Joint Vision 2020 <www-kntwgs-22.ds.boeing.com/reports/OSD-Annual/jv2020b.pdf>.
3. U.S. Government. GIG, Enabling the Joint Vision. Chairman, Joint Chiefs. Jan. 2000 <www.dtic.mil/jcs/j6/enablingjv.pdf>.
4. Capstone Requirements Document, Global Information Grid (GIG). JROCM 134-01, 30 Aug. 2001 <http://xanadu.ds.boeing.com/~moody/docs/grid/GIG_CRD_(Final).pdf>.
5. Internet Protocol. IPv4 and IPv6, RFC 791 and 2460, respectively <www.ietf.org/rfc.html>.
6. U.S. Government. DoD Joint Technical Architecture. Department of Defense <www-jta.itsi.disa.mil>.
7. U.S. Government. Defense Information Infrastructure Common Operating Environment. Defense Information Systems Agency <http://diicoe.disa.mil/coe>.

Note

1. For more information on the consortium, please contact Karen Mowrey at karen.m.mowrey@boeing.com or call (714) 742-2157.

About the Author

Bradley C. Logan is a senior systems engineer with Boeing's Integrated Defense Systems where he works network-centric modeling and simulation efforts along with network infrastructure architecture definition. Previously, Logan worked both commercial and defense sectors in application domains from image and signal processing to reactive control systems. He has a Master of Science in electrical engineering from U.C. Berkeley and a Bachelor of Science in engineering from Harvey Mudd College, Claremont, Calif.

The Boeing Company
Strategic Architecture
Integrated Defense Systems
3370 Miraloma Ave.
MC 031-DB20
Anaheim, CA 92803-3105
Phone: (714) 762-3255
E-mail: bradley.c.logan@boeing.com

New Spreadsheet Tool Helps Determine Minimal Set of Test Parameter Combinations

Gregory T. Daich
Software Technology Support Center

Combinatorial testing is a method for identifying incorrect interactions between various parameters called test factors, usually with a goal to run a minimum number of tests. "Give us your tired, your poor, your huddled … (testers) yearning to breathe free" [1] *from executing endless and senseless combinations of test cases. This article explains how to minimize test parameter combinations using a new spreadsheet tool called ReduceArray2.*

Your manager assigns you a new testing project. "I want you to take over the system integration testing of the Web Time Charging System (TCS). We've got three weeks to get it out the door and we're concerned about the integration of all the Web-TCS components."

The neural cogs in your head start churning. You know the Web TCS must run on your standard Brand X and Brand Y central processing units and the company's current operating systems (OS): Win 98, Win NT, Win 2000, and Win XP. Each of these platforms must support Microsoft Internet Explorer Version 5.5 and 6.0 and Netscape Version 7.0.

Your manager interrupts your daze and says, "And I don't have to remind you about what that last delivered bug cost us."

"What? Oh yeah! I'll get right on it," you profess, while your manager hurries off to another meeting.

The Web TCS has two operational network modes: internal intranet and modem remote. Employees can log their time in both modes. Various default parameters are established depending on the user's type of employee classification, including salaried, hourly, part-time, or contractor. These parameters include default shift, available paid holidays, etc. Also, the user can set the time increment in minutes to six, 10, 15, 30, or 60.

These features and parameters will be combined into various test configurations, however, one key question is, "What is the most effective, smallest set of test configurations that will find the majority of serious parameter interaction defects?" (I'll answer that soon.)

The TCS is defined by five principal use cases[1]. System-level test scenarios will be defined to exercise each use-case in the various test configurations chosen. The use-cases are listed as follows:

1-Login
2-Log Time
3-Submit Time Sheet
4-Maintain Charge Codes
5-Select Time Period

The test group has already defined 15 test scenarios to use to test each test configuration. Test scenarios for the Login use-case include (1) successful login on first attempt, (2) successful login after one failed attempt, and (3) unsuccessful login after three failed attempts. Twelve similar test scenarios were defined for the other four use-cases.

Management has expressed concern about integration defects delivered in recent releases. Any seriously defective interactions between features and various user-assigned and system configuration parameters could prove fatal to the Web TCS upgrade effort and the future of your group (you've heard this before). At any rate, you need to test each parameter paired with every other parameter to be sure that there are no incompatibilities. I will discuss a simple, straightforward approach for obtaining or getting very close to a minimum set of test configurations that the reader will be able to immediately use on his or her project.

This approach or technique will answer the question posed earlier, "What is the most effective, smallest set of test configurations that will find the majority of serious parameter interaction defects?" Notice the qualification "majority of serious … defects." Remember that no amount of testing can find all defects. However, most people accept as self-evident that effective testing techniques can lead to increased confidence and to fewer delivered defects and happier customers. This does not mean that these techniques should displace other effective and efficient means for improving or assuring the quality of the system.

What Have We Got?

Essentially, there are six parameters called test factors that are of most interest from a system integration testing perspective. Table 1 lists the six test factors with their associated options. If all combinations of these factors were tested, that would require the following:

2x4x3x2x4x5 = 960 test configurations

Since each test configuration requires 15 system-level test scenarios, the result is a total of 960 x 15 = 14,400 test scenarios that must be executed. There is not time to execute all 14,400 test-scenarios in three weeks. Say that it takes about three hours to execute the 15 test scenarios for each configuration, which includes setup and reporting. If you consider that there are about six hours per day of productive test execution time, not counting unpaid overtime that you covertly plan to minimize, that gives you 90 hours or 30 test configurations that you have time to perform. Is there hope? Can you test all important combinations of parameters in less than 30 test configurations?

One approach in minimizing the number of test configurations that some organizations use is to test the most common – or important – configuration and then vary one or more parameters for the next test configuration and then test that. Looking at Table 1, you can see that a minimum of five test configurations (rows) are required to test all options at least once. Just look at each row and pick the assigned values. For columns that have dashes, use the preceding value. However, the concern is to test for possible bad interactions between parameters. This method will not be adequate. Madhav Phadke and others have focused on combinatorial testing techniques that arguably "have the highest effectiveness, measured in terms of the number of faults detected per test" [3].

Identifying a minimum set of tests that check each parameter interacting with every other parameter (i.e., all pairs of parameters) is often a very difficult venture if pursued in an ad hoc, non-systematic fashion. Orthogonal arrays (OAs) provide a systematic means for identifying a minimal set of highly effective tests. Unfortunately, some training in combinatorial software testing techniques available in the industry today is not very helpful in teaching this. But before discussing how to use OAs effectively to

find that minimal set, I need to outline a basic fault model of interest when conducting integration testing, and introduce a little terminology.

Basic Fault Model

Jeremy Harrell published a technique that he calls the Orthogonal Array Testing Strategy (OATS) for manually computing a set of tests from published OAs that is very effective [4]. He does a good job of characterizing a basic fault model that is the foundation for using the OATS technique, which is as follows [4]:

- Interactions and integrations are a major source of defects.
- Most … defects are not a result of complex interactions such as, "When the background is blue and the font is Arial and the layout has menus on the right and the images are large and it's a Thursday then the tables don't line up properly." Most of these defects arise from simple pair-wise interactions such as, "When the font is Arial and the menus are on the right the tables don't line up properly."
- With so many possible combinations of components or settings, it is easy to miss one.
- Randomly selecting values to create all of the pair-wise combinations is bound to create inefficient test sets and test sets with random, senseless distribution of values.

Phadke adds to this basic fault model with his discussion about the following testing techniques [3]:

- **One-Factor-at-a-Time Testing**. This method varies one factor at a time and would require more than the minimum five tests mentioned earlier. But this makes it easier to identify the defective parameter. However, this technique does not expect to encounter any bad interactions between the given parameters since it does not attempt to cover all the pairs of parameters. It only finds what Phadke calls single-mode faults.
- **Exhaustive Testing**. For any non-trivial system, this will not be possible. Even if all 960 test configurations were tested, which would find nearly every bad interaction between the given parameters, there will be many more tests with varying circumstances that could be conceived that could take a lifetime and more to conduct.
- **Deductive Analytical Method**. This method attempts to cover all important paths in the code. Any testing strategy should be augmented by some of this type of testing. In fact, this is one type of testing that developers should conduct on their new components prior to integration testing.
- **Random/Intuitive Method**. This is the most common method used by independent test organizations. This method can be very effective at finding defects but the level of coverage is often questionable.
- **Orthogonal Array-Based Testing**. This method finds all double-mode faults that are two parameters conflicting with each other. An example of a double-mode fault is one parameter overshadowing another, inhibiting required processing of that other parameter.

A	B	C	D	E	F
CPU	**OS**	**Browser**	**Network**	**Type of Employee**	**Time Increment**
Brand Y	NT	IE 6.0	Modem	Salaried	6
Brand X	98	IE 5.5	Internal	Hourly	10
--	2000	NS 7.0	--	Part-Time	15
--	XP	--	--	Contractor	30
--	--	--	--	--	60

Table 1: *Test Factors and Options*

Terminology

Some introductory terms for understanding OAs include the following [4]:

- **Orthogonal Array**. Two-dimensional arrays that possess the interesting quality that by choosing any two columns in the array you receive an even distribution of all the pair-wise combinations of values in the array.
- **Runs**. The number of rows that are the potential test configurations or test cases.
- **Factors**. The number of columns that are variables or parameters of interest.
- **Levels**. The number of options or values for each factor.
- **Strength**. The number of columns it takes to see each option equally often.
- **OAs**. These are named OA(N, s^k, t) or OA(N, s^k) if its strength is two. This indicates an OA with N runs, k factors, s levels, and strength t.
- **Mixed Arrays**. These are named MA(N, $s1^{k1}$, $s2^{k2}$, etc.). This indicates a mixed-level, asymmetric OA with N runs, k1 factors at s1 levels, k2 factors at s2 levels, etc., and with strength 2 (assume strength 2 if it is not stated).

Note that the number of runs is dependent on the number of factors, levels, and strength. For example, OA(9,4^3) means you have 9 runs required to cover 4 test factors with 3 options each (see Table 2). Since the strength is assumed 2, this OA covers all pairs of parameters. OA(64,6^4,3) means you have 64 runs required to cover all three-way combinations (strength 3) of six test factors with four options each. Look these up on the Web or in a book on statistics[2].

A Powerful Technique

It is not easy to create non-trivial OAs, which are OAs with more than three factors. Furthermore, some currently available automated tools that produce sets of tests covering all pairs of parameters do not create actual OAs or a minimum set of tests. After all, it is a very difficult, discrete mathematics problem to create OAs. These tools may come fairly close to a minimum set but if you can save yourself from having to run even one more test, you are going to want to find that minimal set, especially if it does not require investing any more time. There are tools that can compute OAs or all pair combinations resulting in a minimal set of tests for all reasonable numbers of factors and options but they may cost much more than you may want to spend[3].

The OATS technique for manually generating a minimal or near minimal set of tests is actually better than some tools I have seen. In other words it produces a smaller set of tests that exercise all pair-wise combinations of parameters. OATS is simple and straightforward and should be used by a lot more organizations to help with software and system integration testing efforts. Briefly, it includes the following steps [4]:

1. Decide the number factors to test.
2. Decide which options to test for each factor.
3. Find a suitable OA with the smallest number of runs to cover all factors and options.

Table 2: *OA(9,4^3)*

	A	B	C	D
1	0	0	0	0
2	0	1	1	2
3	0	2	2	1
4	1	0	1	1
5	1	1	2	0
6	1	2	0	2
7	2	0	2	2
8	2	1	0	1
9	2	2	1	0

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Array					TP	54	Results						
2	A	B	C	D		TC#	RP	UP	A:B	A:C	A:D	B:C	B:D	C:D
3	0	0	0	0		1	6	6	1	1	1	1	1	1
4	0	1	1	2		2	6	6	1	1	1	1	1	1
5	0	2	2	1		3	6	6	1	1	1	1	1	1
6	1	0	1	1		4	6	6	1	1	1	1	1	1
7	1	1	2	0		5	6	6	1	1	1	1	1	1
8	1	2	0	2		6	6	6	1	1	1	1	1	1
9	2	0	2	2		7	6	6	1	1	1	1	1	1
10	2	1	0	1		8	6	6	1	1	1	1	1	1
11	2	2	1	0		9	6	6	1	1	1	1	1	1
12						SP	54							

TP = Total Pairs (54)
UP = Unique Pairs (Six on Each Row)
TC# = Test Case Number
RP = Number of New Pairs on a Row (Six on Each Row)
A:B = Column A Paired With Column B, etc.
SP = Sum of Pairs (54)

Table 3: *Results of Analyzing OA(9,4³)*

4. Map the factors and options onto the array.
5. Choose values for any options that are not needed (call them *leftovers*) from the valid remaining options and delete any columns (factors) that are not needed. A given test situation may not need all the factors or all the options that a particular OA provides.
6. Transcribe the runs into test cases, adding additional combinations as needed.

The selection of which OA to use can pose a bit of a challenge. The good news is that you do not generally need to compute a new OA for many test situations. As the technique discusses, if there is not a specific OA for your test situation, you can use one that is similar, delete extra factors (columns), and choose values for the extra options consistent with your test situation. In other words, the real work in creating the arrays has already been done. And it only takes a few minutes to apply the array to a specific test situation. That is pretty powerful, but not enough people know about it.

Techniques vs. Tools

When using the OATS technique to identify a minimal set of tests, you may encounter an OA where each pair appears more than once but the same number of times (evenly). This may be considered overkill from a software testing perspective since all you usually want is to test each pair once. However, identifying the actual minimum set and eliminating any overkill is generally difficult and time consuming without the aid of a tool to quickly and easily see the pair combination counts. Thus, I have prepared an Excel spreadsheet tool called ReduceArray2 that computes the total number of pairs of parameters possible based on the array's factors and options, and it identifies any missing pairs[4]. It counts the number of unique pairs in the array for each row from top to bottom so you can compare it with the total number of pairs. It also computes the number of occurrences of each pair in the array and displays them in the spreadsheet in a concise and easily analyzed form.

Table 3 shows the results of analyzing OA(9,4³) using the ReduceArray2 tool. The top row identifies the columns in the spreadsheet. The left column identifies the rows in the spreadsheet. This makes it fairly simple to identify any extra test cases so that rows can be deleted to reduce the size of the array. Extra test cases are rows with no unique pairs. Note that every pair in OA(9,4³) is unique, thus every one is needed. However, the OATS technique will often have values that are not needed (called leftovers) that will create redundant pairs that can be rearranged to create rows with no unique pairs that can then be deleted.

In order to identify a set of tests for the factors and options in Table 1 using the OATS technique, you could use OA(25,5⁶). This would result in 25 tests. Additional rows could be deleted after rearranging some pairs but that would require some fairly labor-intensive study and manual effort without tool support. Also, using one automated tool with which I am familiar produced 26 tests for the test situation in Table 1.

Using ReduceArray2 to assist in finding extra tests, I was able to reduce the set to 20, which is the minimum number of configurations to test all pairs for this situation. The minimum count of 20 was derived from the two factors with the most options. The Time Increment factor has five options and the OS factor has four options. Thus, we know that there must be at least 5 x 4 = 20 runs to cover all combinations of those two options. The trick is to cover all the other pair combinations in those 20 runs. With the visibility provided by the ReduceArray2 tool, this became a much easier task.

Table 4: *Subset Test Situation*

A	C	D	E
CPU	Browser	Network	Type of Employee
Brand X	IE 5.5	Internal	Salaried
Brand Y	NS 7.0	Modem	Hourly
--	--	--	Part-Time

Table 5: *Subset Test Situation With Leftovers Highlighted*

	A	B	C	D
1	Array			
2	A	C	D	E
3	0	0	0	0
4	0	1	1	2
5	0	~~2~~ 0	~~2~~ 0	1
6	1	0	1	1
7	1	1	~~2~~ 1	0
8	1	~~2~~ 1	0	2
9	~~2~~ 0	0	~~2~~ 0	2
10	~~2~~ 1	1	0	1
11	~~2~~ 0	~~2~~ 0	1	0

Table 6: *Subset Test Situation With Leftovers Assigned*

	A	B	C	D
1	Array			
2	A	C	D	E
3	0	0	0	0
4	0	1	1	2
5	0	0	0	1
6	1	0	1	1
7	1	1	1	0
8	1	1	0	2
9	0	0	1	2
10	1	1	0	1
11	0	0	1	0

Demonstration

The following uses the OATS technique augmented with a few extra steps to identify a minimal set of tests to cover all pairs. To simplify the demonstration and reduce the number of tests, only look at the test factors and options A, C, D, and E in Table 4. The following lists each OATS step with our actions in italics:

1. Decide the number of factors to test. *We chose the four test factors in Table 4.*
2. Decide which options to test for each factor. *The options for test factors A, C, D, and E are listed in Table 4.*
3. Find a suitable OA with the smallest number of runs to cover all factors and options. *A suitable OA was selected that is OA(9,4³), see Table 2. A suitable array has at least the number of factors and options with*

a minimum of leftovers. In this case, there are no leftover factors (columns) and factors A, C, and D each have one leftover option.

4. Map the factors and options onto the array. *Table 5 identifies the leftovers that are the highlighted boxes.*
5. Choose values for any leftovers from the valid remaining options and delete any columns that are not needed. *Harrell* [4] *rightly suggests that we assign alternating values for each factor as shown in Table 5. The array with the newly assigned options is shown in Table 6.*
   5a. (Extra step) Delete any rows that have no unique pairs. *Table 7 shows the results of analyzing our subset test situation. Spreadsheet row 11 contains no new pairs in the row (see under RP heading). This row can be deleted. After deleting row 11, reanalyze the array to produce Table 8.*
   5b. (Extra step) Rearrange cell values to create a row with no unique pairs. If this cannot be done, then quit; otherwise, go back to step 5a. *Row 10 column M shows that the B:D columns pair is unique (only one occurrence). Move that pair combination to row five by assigning the value one to B5 (note that this is the spreadsheet cell location) as shown in Table 8. The results of reanalysis after reassigning cell B5 are shown in Table 9. Continuing steps 5a and 5b produces Table 10 with six resulting test cases.*
6. Transcribe the runs into test cases, adding combinations as needed. *Table 11 (see page 30) shows the transcribed values that define the configurations for testing interactions between test factors A, C, D, and E in Table 4.*

"Wait a minute!" you say. "Steps 5a and 5b are easier said than done." You are right, especially if you have more factors! The ReduceArray2 tool displays the number of unique pairs automatically in each row so that you can identify any rows that are not adding new pairs of parameters. They are, in other words, overkill, so you can delete them. After deleting a row, you need to recompute the pair counts so you can identify other rows with no unique pairs that can be deleted. If no rows can be deleted, then you would try to rearrange cell values to create a row with no unique pairs.

Further Automation

Now that I have described the OATS technique and my extensions to further reduce the number of test combinations using the ReduceArray2 tool, I have some more good news. ReduceArray2 also does steps 5a and 5b automatically using its embedded ReduceArray2 macro. In other words, do steps one through five and then use the ReduceArray2 macro to automatically rearrange parameter values and delete extra rows without losing any pair combinations.

ReduceArray2 provides a near minimal set of tests using simple one-cell-at-a-time rearrangements. If you start with a better initial arrangement, you may be able to reduce it further. ReduceArray2 saves a lot of time and effort and will find a minimum set for some configurations. Furthermore, you can define specific combinations that are required and specific combinations that are to be excluded. I would be happy to walk you through a demonstration on this. Another macro called Names automatically transcribes the OA values to the names in Table 11, page 30.

Epilogue

Go back to the original test situation. Using ReduceArray2, you can create a set of 20 test configurations from the test factors and options in Table 1. You have time to run 15 test scenarios in each of the 20 test config-

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Array					TP	30	Result						
2	A	C	D	E		TC#	RP	UP	A:B	A:C	A:D	B:C	B:D	C:D
3	0	0	0	0		1	6	1	4	3	2	3	2	1
4	0	1	1	2		2	6	2	1	2	2	2	2	1
5	0	0	0	1		3	3	1	4	3	1	3	2	2
6	1	0	1	1		4	5	2	1	2	2	2	2	1
7	1	1	1	0		5	4	2	3	2	1	2	1	2
8	1	1	0	2		6	4	1	3	2	1	2	2	2
9	0	0	0	2		7	1	1	4	3	2	3	1	2
10	1	1	0	1		8	1	1	3	2	2	2	1	2
11	0	0	1	0		9	0	0	4	2	2	2	2	2
12						SP	30							

Note: Cells with *X* can be any valid value (called "do not cares")

Table 7: *Subset Test Situation Results of Analysis*

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Array					TP	30	Result						
2	A	C	D	E		TC#	RP	UP	A:B	A:C	A:D	B:C	B:D	C:D
3	0	0	0	0		1	6	3	3	3	1	3	1	1
4	0	1	1	2		2	6	3	1	1	2	2	2	1
5	0	0	0	1		3	3	1	3	3	1	3	2	2
6	1	0	1	1		4	5	3	1	2	2	1	2	1
7	1	1	1	0		5	4	3	3	2	1	2	1	1
8	1	1	0	2		6	4	1	3	2	1	2	2	2
9	0	0	0	2		7	1	1	3	3	2	3	1	2
10	1	1	0	1		8	1	1	3	2	2	2	1	2
11						SP	30							

Table 8: *Subset Test Situation Reanalysis After Deleting a Row*

Table 9: *Subset Test Situation Reanalysis After Reassigning Cell B5*

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Array					TP	30	Result						
2	A	C	D	E		TC#	RP	UP	A:B	A:C	A:D	B:C	B:D	C:D
3	0	0	0	0		1	6	3	2	3	1	2	1	1
4	0	1	1	2		2	6	2	2	1	2	2	2	1
5	0	1	0	1		3	4	1	2	3	1	3	2	2
6	1	0	1	1		4	6	4	1	2	2	1	1	1
7	1	1	1	0		5	4	3	3	2	1	2	1	1
8	1	1	0	2		6	3	1	3	2	1	3	2	2
9	0	0	0	2		7	1	1	2	3	2	2	1	2
10	1	1	0	1		8	0	0	3	2	2	3	2	2
11						SP	30							

Table 10: *Final Results Six Test Cases*

	A	B	C	D	E	F	G	H	I	J	K	L	M	N
1	Array					TP	30	Result						
2	A	C	D	E		TC#	RP	UP	A:B	A:C	A:D	B:C	B:D	C:D
3	0	0	0	0		1	6	4	2	2	1	1	1	1
4	0	0	1	2		2	5	4	2	1	1	2	1	1
5	1	0	1	1		3	5	4	1	2	1	2	1	1
6	1	1	1	0		4	5	4	2	2	1	1	1	1
7	1	1	0	2		5	5	4	2	1	1	2	1	1
8	0	1	0	1		6	4	4	1	2	1	2	1	1
9						SP	30							

A	C	D	E
CPU	Browser	Network	Type of Employee
Brand X	IE 5.5	Internal	Salaried
Brand X	IE 5.5	Modem	Part-Time
Brand Y	IE 5.5	Modem	Hourly
Brand Y	NS 7.0	Modem	Salaried
Brand Y	NS 7.0	Internal	Part-Time
Brand X	NS 7.0	Internal	Hourly

Table 11: *Transcribed Options*

urations within the timeframe mandated by management. And you even have time to rerun some tests if you encounter some defects and must retest the system.

By the way, I meant no disrespect in this article's preface to the poem written by Emma Lazarus enshrined on a plaque near the Statue of Liberty. But since I have seen literally thousands of tired testers yearning for more effective and more efficient test techniques in my software test-consulting career, I thought I would borrow the theme.

The test situation in this article was contrived. Details about actual feature requirements were ignored to simplify the discussion. But it does make the point that, when you know the features and parameters that could possibly interact incorrectly with each other, then you can follow a simple systematic approach to identify a minimal or near minimal set of tests to test all parameter pairings. That is a powerful capability. You can still pick and choose which test runs to ignore and which additional ones to add. However, identifying the test factors and options in the first place is often very difficult. More research is needed in this area.

It is interesting to note that Phadke's perspective back in 1997 was that,

> … the number of tests needed for (the OA testing) method is similar to the number of tests needed for the one-factor-at-a-time method, and with a proper software tool (likely his Robust Testing Method tool), the effort to generate the test plan can be small. [3]

Harrell's article [4] and this article show you how to create tests without expensive tools; if you have Internet access, download some OAs. Also, I have shown you how to augment the OATS technique with a few additional steps to reduce the number of tests for many types of test situations where you have leftovers. This method also works when the OA has multiple occurrences of pairs that are common when the number of factors is higher. Of course, the Reduce Array2 tool makes it even easier to do this.

Whether you are following formal practices, using defined processes advocated by the Software Engineering Institute's Capability Maturity Model®, or you are applying agile exploratory testing [5] methods, this combinatorial testing technique will certainly help you obtain better integration test coverage. When you are concerned about various features and parameters interacting incorrectly with each other, use this OATS technique augmented with the ReduceArray2 tool or purchase and use a tool such as Automatic Efficient Test Generation. It may not be worth the risk of letting those defects get delivered to your customer.◆

References

1. Lazarus, Emma. "The New Colossus." Statue of Liberty plaque.
2. Rational Unified Process, 2001, Use Case Template.
3. Phadke, Madhav S. "Planning Efficient Software Tests." CrossTalk Oct. 1997.
4. Harrell, Jeremy M. "Orthogonal Array Testing Strategy (OATS) Technique." Seilevel, 2001 <www.seilevel.com/OATS.html>.
5. Bach, James. "Exploratory Testing Explained." Ver. 1.1. Satisfice, Inc., 19 Jan. 2003 <www.satisfice.com>.

Notes

1. A use case is a description of a sequence of actions that a system performs that yields an observable result of value to a particular actor (user) [2].
2. See Sloane, N. J. A. "A Library of Orthogonal Arrays" <www.research.att.com/~njas/oadir>. Also see Sherwood, George. "On the Construction of Orthogonal Arrays and Covering Arrays Using Permutation Groups" <http://home.att.net/~gsherwood/cover. htm>.
3. See the Automatic Efficient Test Generation System by Telecordia Technologies at <http://aetgweb2.argreenhouse.com>.
4. Available at no cost at <www.stsc.hill.af.mil>.

The Software Technology Support Center's No-Cost Service Offer

If you will identify the factors and options for your particular system, the Software Technology Support Center (STSC) can either help you to identify a minimal or near minimal set of tests to test all pair-wise combinations, or the STSC can identify these for you. Information can be easily received by e-mailing a spreadsheet formatted like Table 1 to the STSC HelpDesk. This is a no-cost service offer to Department of Defense (DoD) organizations. Go to <www.stsc.hill.af.mil> for more information. Of course, non-DoD organizations are welcome to inquire also. We have seen the development costs for many organizations reduced because of the information we have shared with DoD organizations and their contractors.

About the Author

Gregory T. Daich is a senior software engineer with Science Applications International Corporation currently on contract with the Software Technology Support Center (STSC). He supports STSC's Software Quality and Test Group with more than 26 years of experience in developing and testing software. Daich has taught more than 100 public and on-site seminars involving software testing, document reviews, and process improvement. He consults with government and commercial organizations on improving the effectiveness and efficiency of software quality practices. He has a master's degree in computer science from the University of Utah.

Software Technology Support Center
OO-ALC/MASEA
6022 Fir Ave.
Bldg. 1238
Hill AFB, UT 84056-5820
Phone: (801) 777-7172
E-mail: greg.daich@hill.af.mil

Softbucks

As the software industry continues to mature we often look for examples or analogies from industries that have experienced similar evolution. While studying the coffee industry as a commodity investment, I found some interesting similarities between the coffee and software industries.

Old Brew, New Brew, or Strange Brew

The coffee industry started by shipping ground coffee beans in cans to home and restaurant percolators. Customers chose between Juan Valdez and Maxwell House's "good to the last drop" coffees. It was all about volume.

Coffee epicureans emerged out of the search for the perfect cup of joe. Not trusting peasant production, epicureans bought beans, roasters, grinders, and high-end percolators. This new market was about margins – making a lot of money on a smaller group of affluent buyers.

Out of the northwest came the pretentious coffee crowd. This new market had little to do with coffee and more to do with where and with whom you drank your coffee. Suppliers integrated bean, brew, backdrop, and boutique to create ambiance. It was about selling an experience.

Instead of forcing a single market delivery system, coffee suppliers grew markets around their customer's lifestyles and desires, strengthening the industry as a whole. This made room for more industry participants and serviced percolators, epicureans, and the pretentious alike.

The software industry materialized from the need to tame monolithic computing machines. Solutions, like our patrons, were simple and similar. The market was seen with an eye single to software – one problem, one method, one process, one technique, all solved by the Holy Grail of software. During the industry's myopic search, its customers have matured with a plethora of software uses and desires.

We can learn from the coffee industry's divide-and-conquer strategy. Divide and conquer, found in algorithms like Heapify, Merge Sort, Quicksort, and Fast Fourier Transforms, yields elegant, simple, and often efficient solutions. It allows software suppliers, like coffee suppliers, to compliment customer lifestyles and desires, and strengthen the market.

This is not requirements management – determining what a customer wants – but macro-requirements management: determining what a market wants and will bear. It requires focus on specific domains and tailoring processes, skills, and techniques to serve each market. It's a best-of-breeds versus best-of-show approach.

Available vs. Fresh

How do restaurants balance the desire for quick and fresh coffee? Do they brew a lot of coffee that gets stale or make customers wait for each fresh pot? The solution was to place a one-third-full line on each pot. When the coffee reaches the line, a new pot would be started and available when the current pot ran out. A simple process change resolved the competing requirements of quick and fresh.

Software customers also want their applications quick and fresh (latest features). The software industry offers speed or quality but seldom both. Let's take a cue from coffee servers and focus on effective, memorable, and easy-to-implement processes rather than defining processes ad infinitum. The complexity of serving coffee does not compare to software development but in reality software is developed using process habits versus a process cookbook.

Good Beans

With coffee, taste begins with the bean. Delivery processes make little difference if you have bad beans. To deliver quality coffee, you have to start with quality beans.

Quality software begins with good people. A good process implemented by inept managers, engineers, and personnel will produce poor software every time. A poor process, while limited in capability, can be overcome by a good team. In fact, in the long run a good team leads to the development of good processes. More time, resources, and effort should be spent on hiring, retaining, and motivating first-class software teams.

Don't Forget the Caffeine

Coffee has taken many forms like espresso, latte, mocha, cappuccino, and frappuccino. Customer wants and needs have led to variations on a basic way to deliver the energizing kick of caffeine. Extracting and prioritizing customer wants and needs helps to develop a good coffee menu.

Extracting and prioritizing software requirements lead to customer satisfaction. While trivialized, customer requirements extraction can be extremely difficult. Prioritizing requirements can be hard and tying them together even harder. Like coffee's basic premise of delivering the energizing kick of caffeine, software's basic premise is effective automation and one should not stray from that principle. Always ask yourself how are you making your customer's life easier.

Would You Like Coffee With Your Condiments?

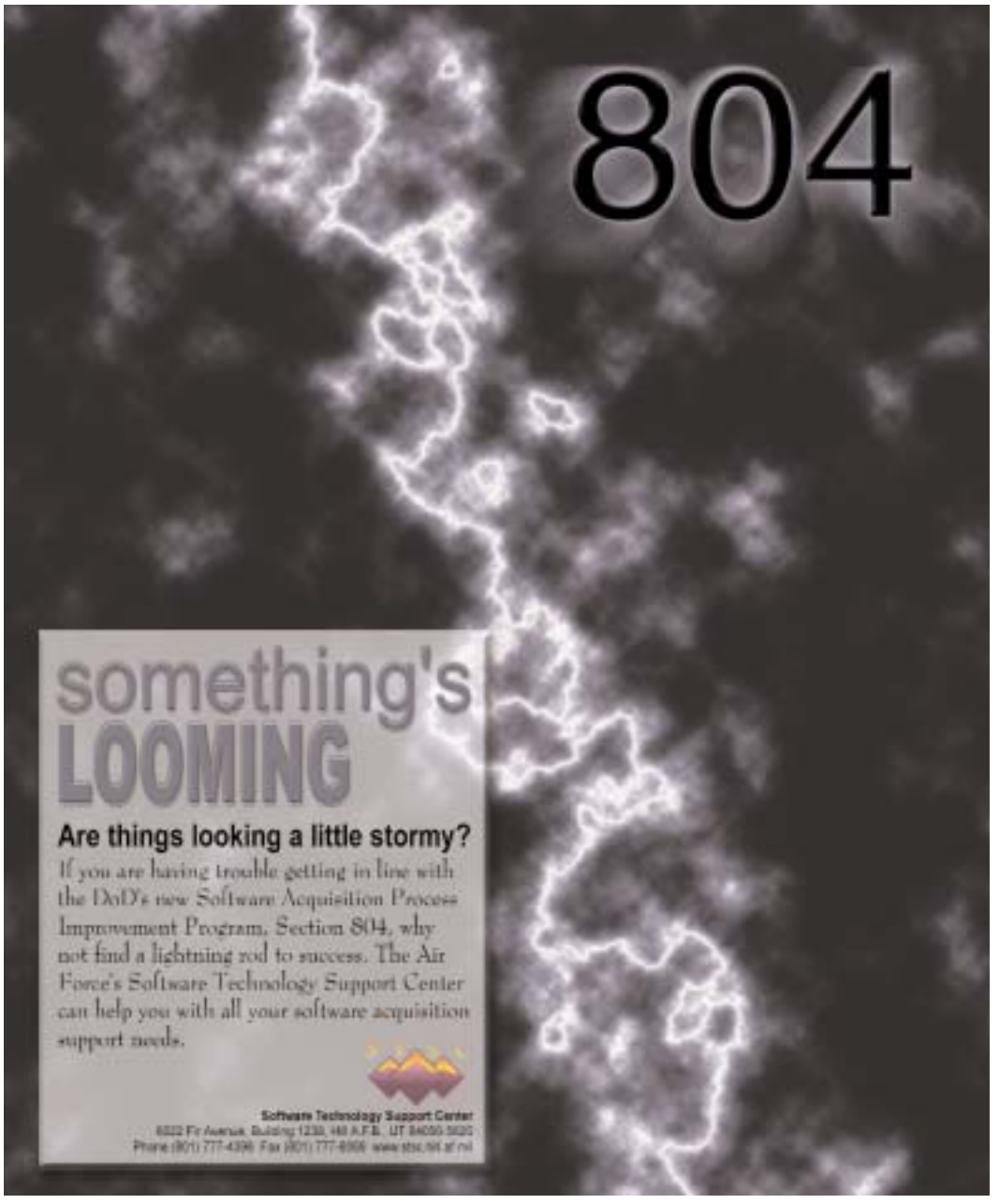
Watching modern day coffee preparation is like watching Dustin Hoffman in the movie "Rain Man" preparing for the Wheel of Fortune. It's all about ritual, timing, and the condiments. Adding cream, sugar, steam, ice, whipped cream, and flavorings it's often hard to find the coffee for the condiment. What happened to "give it to me black?"

It's time to face the fact that software, an enabling technology, is more a condiment than main course. It is useless in and of itself but makes that which it enables better. Software should develop its own variation of BASF's marketing slogan: "We don't make a lot of the products you buy. We make a lot of the products you buy better®." In our industry, we don't make the computer; we make the computer interactive. We don't make the network; we make the network communicate. We don't make the bombs; we make the bombs smart. We don't make the espresso; we make the espresso machine accurate.

Let's strive to make software good to the last bit.

– Gary Petersen
Shim Enterprise, Inc.

® BASF slogan is registered in the U.S. Patent and Trademark Office.



Sponsored by the Computer Resources Support Improvement Program (CRSIP)

Published by the Software Technology Support Center (STSC)

CrossTalk / MASE
6022 Fir Ave.
Bldg. 1238
Hill AFB, UT 84056-5820

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