CROSSTALK

September 2004 The Journal of Defense Software Engineering Vol. 17 No. 9

THE SOFTWARE EDGE:
ENABLING THE WARFIGHTER

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
SEP 2004		**00-00-2004 to 00-00-2004**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
CrossTalk. The Journal of Defense Software Engineering. Volume 17, Number 9, September 2004	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **517 SMXS MXDEA,6022 Fir Ave,Hill AFB,UT,84056-5820**	8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

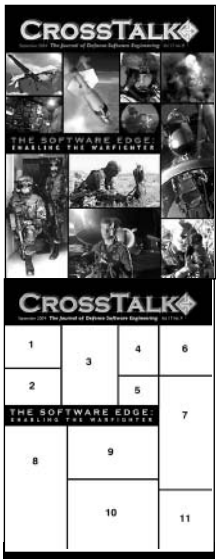
16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**32**	

The Software Edge

Software Engineering Technology

Open Forum

Departments



ON THE COVER

Cover Design by Kent Bingham.
Cover Photo © Image 100 Ltd.

1. MQ-1 Predator. 2. B-52 radar navigator. 3. Joint Direct Attack Munition, photo © Boeing. 4. AH-64D Apache Longbow Helicopter pilot. 5. A Laser highlights a target during a weapons interdiction mission in Iraq. 6. A BGM-109 Tomahawk cruise missile warhead detonates. 7. U.S. Air Force F-15A Eagle pilot. 8. New Land Warrior components. 9. End-User Terminals field test. 10. U.S. Marine Harrier pilot during a SOCEX workup. 11. AN/PAQ-1 laser target designator exercise.

For more detailed information on this map, visit <www.stsc.hill.af.mil/crosstalk>.

CROSSTALK

PUBLISHER	*Tracy Stauder*
ASSOCIATE PUBLISHER	*Elizabeth Starrett*
MANAGING EDITOR	*Pamela Palmer*
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

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail or use the form on p. 29.

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: stsc.webmaster@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.

Greater Combat Effectiveness

I recently attended a briefing given by an F-16 pilot who had flown many missions in Operation Iraqi Freedom. He could not have been more complimentary of the group of software managers and engineers that he was addressing. As he mentioned numerous times, he flies the planes but it is the engineers who are designing and coding the software that in turn enables him to do his job better and put "bombs on target."

As a software engineer, it is easy to get immersed in the nitty-gritty programming aspects of your job and lose sight of the bigger picture. In this month's issue, we highlight the bigger picture of how software plays an ever-increasing role in the U.S. military's combat effectiveness.

In *Software Wars* by Susan Weaver, we begin with a look at how software has evolved to become a key enabler for the Navy's dual-role F/A-18 Hornet aircraft. This article describes the increased capability made possible through the software of onboard systems such as the dual-role radar, heads-up and heads-down displays, weapon delivery systems, and the avionic's digital multiplex bus architecture.

Next, in *Tomahawk Cruise Missile Control: Providing the Right Tools to the Warfighter* by Marcus Urioste, the Tactical Tomahawk Weapon Control System (TTWCS) is described. The TTWCS program enables a reduction in the Tomahawk timeline by placing the missile's mission planning function aboard the firing unit. This article discusses this major software-based reengineering upgrade program aimed at bringing more capability to officers and sailors onboard U.S. surface ships and fast attack submarines.

Software also plays a key role in the move toward a joint net-centric warfare capability. As an information infrastructure, the Global Information Grid (GIG) will improve data routing and shared situational awareness. In *Service-Oriented Architecture and the C4ISR Framework*, Dr. Yun-Tung Lau presents a modeling approach that has been applied to the architecture development of Net-Centric Enterprise Services (NCES). The NCES provides the core enterprise services supporting various communities of interest to the GIG.

In our Software Engineering Technology section, we bring you three articles that describe technology advancements to further sharpen the software edge. First, in *Executable Specifications: Language and Applications* by Dr. Doron Drusinsky and Dr. J.L. Fobes, a formal method of verification is described that can be applied to requirements simulation before software implementation, as well as to a variety of other defense applications to ensure safety and security. Second, in *Executable and Translatable UML* by Stephen J. Mellor, learn the fundamental ideas behind Executable and Translatable UML and how it works in practice to accelerate development and improve the quality of systems. Many senior level managers rely on project managers to present pertinent measurement data that enables the decisions they make. In *What You Don't Know Can Hurt You*, the third article in this section, author Douglas A. Ebert provides helpful questions for senior managers to ask their project managers to ensure the proper set of metrics is being collected for them to act upon.

Finally, *Identifying Essential Technologies for Network-Centric Warfare* by David Schaar is our Open Forum article that shares this author's opinion and research on network-centric warfare (NCW). Schaar discusses NCW from its concept definition to its role in the battlespace to the technologies needed to enable concepts, including better awareness of the enemy and friendly forces.

As shown in this set of articles, it is clear that software plays a big role in increasing the warfighter's combat effectiveness. The F-16 pilot I listened to showed obvious joy in describing what software brings to his job today. I can only imagine what he might be saying about its impact 10 years from now.

Tracy L. Stauder

Tracy L. Stauder
Publisher

Software Wars

Susan Weaver
L-3 Communications Government Services, Inc.

The basic premise behind the F/A-18 aircraft design was that capability growth could be achieved through software upgrades rather than requiring frequent hardware changes to increase functionality. At least that was the vision back in the late 1970s when the concept for the F/A-18 aircraft was first developed. The F/A-18 platform's recent return to the Persian Gulf region has provided us with a rare opportunity to examine the extent to which we have been able to achieve capability growth through software upgrades, and to consider the lessons we have learned as a major military system.

The U.S. Navy's F/A-18 Hornet is a single- and two-seat, twin engine, multimission fighter/attack aircraft that can operate from either aircraft carriers or land bases [1]. The F/A-18 relies on two primary mission computers for navigational control and weapons employment. The original F/A-18 A/B model aircraft mission computers use 1970s software technology with 32 thousand (K) of memory in each computer. From the start, the F/A-18 Hornet was designed to perform both the fighter and attacker roles. To support these dual roles, multi-function *programmable* radar was created. Instead of being wired to do just one job, the radar would be software reprogrammable.

A programmable radar signal processor and data processor provide the flexibility to change missions and select radar mode based on pilot inputs. For example, the F/A-18 uses the same radar for air-to-air target acquisition and tracking, as well as air-to-ground Doppler beam sharpened target mapping. Another key element was breakthrough technology in human factors and the pilot-to-vehicle interface allowing an effective one-man operation.

A concern in designing a dual role aircraft was how to provide all the information the pilot needed to effectively perform the mission without overwhelming the pilot with information. An innovation incorporated into the Hornet for one-man operation is the combination of the heads-up display (HUD) and three *heads-down* displays. As the pilot looks through the HUD toward the sky, the land, or the sea, dynamic symbols are presented displaying everything the pilot needs to safely fly the plane and deliver weapons. The HUD displays symbols that are projected at infinity, eliminating the need for the pilot's eyes to refocus from long distance, or infinity, to the instrument panel, which reduces incidences of vertigo.

Twenty buttons used to manipulate information presented on the screen surround each heads-down display. By pushing one button, the pilot can see a *menu* of available choices. Then, by pushing additional buttons, a diagram of the weapons being carried is displayed. The pilot selects a weapon for use and enters a delivery program (e.g., number of weapons to be released, release interval, fusing options).

Two of the three heads-down displays are identical, allowing one display to back up the others in a malfunction. The third display can project a moving groundmap combined with additional situational awareness or navigational data. This map is programmed to follow aircraft movement showing aircraft position relative to specific ground features (e.g., roads, railroad tracks, cities).

> *"The efficiency and effectiveness of the weapons were increased through minor upgrades to software. Our ability expanded from delivering multiple weapons on a single target to delivering multiple weapons on multiple targets."*

The HUD and three heads-down displays work in concert via their software programming. For example, air-to-air targets may be displayed from a *bird's eye* view on one of the heads-down displays, while the HUD displays a line-of-sight cue (from the pilot's viewpoint) outlining the highest priority target and current weapon selected. In all cases, critical information on the heads-down display screens also appears via the HUD, usually in a graphical format designed for rapid pilot comprehension.

The center instrument panel contains the up-front-control panel. The aircrew uses the 10-digit keypad panel to change radio frequencies or enter data such as target latitude/longitude. A pilot sometimes changes radio frequencies as many as 40 times an hour. After a little practice, this placement allows the pilot to change frequencies without looking.

During a dogfight maneuver, push-button selection may be difficult due to the number of G-forces being endured by the pilot. Therefore, all controls needed to manipulate the Hornet during stressful maneuvers are located on the engine throttles and the flight control stick. This convention is termed *hands-on throttle and stick*. This enables the pilot to select radar modes, weapons, and targets and control engine power, all with the touch of a finger. Designers of the F/A-18 knew a pilot's survival might depend on a swift response in dealing with attacks from a hostile fighter.

Deployment

The first real test of the F/A-18 A/B came in 1986 with air strikes against Libya. An F/A-18 aircraft attached to the U.S.S. Coral Sea launched high-speed anti-radiation missiles against Libyan air defense radars and missile sites, effectively silencing them during the attacks on Benghazi facilities [1]. After the attack, the F/A-18s were armed and ready to counter any air-to-air or air-to-ground threat the Libyans may have planned.

During this timeframe, the entire aircraft contained approximately one million lines of code. Early F/A-18s contributed in multiple areas, primarily the air defense role in the form of combat air patrol and suppression of enemy air defense. Like all later model F/A-18s, all necessary information to land on the aircraft carrier is available on the HUD. This eases the effort required by the aircrew to *get aboard* and

land safely on the carrier.

Operation Desert Storm

In 1987, the next major variant of the Hornet (F/A-18 C/D) was released to the fleet. The early F/A-18 C/D models utilized 256K processing power in each of its primary computers, with later model C/D aircraft possessing 2,112K memory for each primary processor, and a total of six million lines of code in the aircraft.

To understand the significance of the F/A-18 C/D's dual role, lone operator, and system capabilities, it is useful to imagine what it is like for a pilot to bomb an enemy target. The following is Navy Lt. Nick Mongillo and Lt. Cmdr. Mark Fox's recollection of an event that occurred during Desert Storm.

> Fox and Mongillo had launched their first combat mission. Carrying four 2,000-pound bombs, two AIM-9 heat-seeking Sidewinder missiles, two AIM-7 Sparrow radar-guided missiles, and a centerline 330 gallon external fuel tank, the F/A-18s made their way toward the designated target approximately 550 miles from the carrier.
>
> As they approached the target area, the pilots had their radar in air-to-ground mode when they suddenly received word of approaching enemy fighters. "The E-2 (early warning aircraft) gave us a call saying, 'Bandits on nose at 15,' which is a confirmed bad guy at 15 miles," recalled Fox. "We quickly went back to our radar search mode, got locks on them, confirmed they were bad and shot them both down."
>
> Fox fired two shots to down the first MiG: a Sidewinder followed by a Sparrow. Both missiles hit the Iraqi jet, and the exploding MiG was clearly visible on the videotape that recorded the action through the HUD. "I fired a Sidewinder at what seemed to be a relatively long range, but it wound up working. I wasn't sure it was going to do that, so I fired a Sparrow to make sure," said Fox.
>
> Mongillo's kill came only a few seconds after Fox's and was made at close range with a Sparrow. It was also clearly visible on the HUD videotape. Furthermore, the tapes show two retreating MiGs were within missile range when the Hornets disengaged to complete their bombing run. Both Hornets then dropped their bombs on target from an altitude of 18,850 feet. "The idea of a strike fighter is valid. I'm not going to make any grandiose claims, but I do believe we're the first guys to kill anybody while carrying 8,000 pounds of bombs," said Fox. [2]

Overall, more than 210 U.S. Navy, Marine Corps, and Canadian F/A-18 Hornets were engaged in Operation Desert Storm. More than 6,000 targets were hit by Hornet aircraft flying a variety of missions from fleet air defense to reconnaissance to suppression of enemy air defenses to neutralizing ground forces. The F/A-18 aircraft delivered 18 million pounds of ordnance, and clocked more than 30,000 flight hours while flying 11,000 sorties – an average of 1.2 sorties per day. Throughout Desert Storm, the aircraft averaged 90 percent readiness. Hornets completed 95 percent of scheduled sorties and missed none for maintenance reasons [3].

Desert Storm was the first major conflict to make use of *smart* software-enabled bombs like the Joint Standoff Weapon (JSOW). Smart weapons can be programmed before aircraft takeoff to execute a series of software-controlled navigational changes. These navigational changes minimize the risk of the weapon being disabled by enemy fire before reaching its assigned target. Maverick anti-tank missile and laser-guided bombs developed during the Vietnam War continue to be very effective against high-value targets.

The heart of the F/A-18 is the integration of software and hardware into a single system. During Desert Storm, F/A-18 software was viewed as the glue that held the hardware elements together. Because of this tightly integrated system, changes to the software in one or more of its computers can effect a major improvement in the aircraft's capabilities without requiring hardware or airframe modifications to the aircraft. For example, adding a new weapons system to the aircraft inventory rarely requires hardware modifications. Changing the software is much easier and faster than changing the hardware, shortening the time required to integrate the weapon on the aircraft.

A basic hardware infrastructure is, however, necessary to provide the ability to take advantage of the F/A-18's reprogrammability. A standard language for communicating between processors allows new weapons to be added to the inventory without requiring a hardware change to the aircraft. Just as faster processor upgrades are necessary to support expansion of software applications, the F/A-18 transitioned to faster processors so that *smart* weapons can exchange more complex data in less time.

Operation Iraqi Freedom

As good as F/A-18 operations were in Desert Storm, there was plenty of room for improvement. As of May 1999, Hornet pilots had accumulated more than 3.7 million flight hours and, in the process, established new records daily in safety, reliability, maintainability, and mission performance [1].

Primarily due to multiple software updates, the aircraft in Operation Iraqi Freedom have significantly more combat capability. The most dramatic is the introduction of a family of new global positioning system (GPS)-guided munitions. The Joint Direct Attack Munition (JDAM) and JSOW allow autonomous and highly accurate weapons delivery in all types of weather. GPS-guided weapons improve the lethality of the F/A-18 weapon system over the laser-guided and ballistic weapons used in Desert Storm, which required clear air to be effective.

Additionally, there was a host of other weapon improvements. Some improvements were made within the weapons' software and others within the F/A-18's processors to enhance data exchange between the weapon and aircraft.

The F/A-18 aircraft's source lines of code now exceed 8.3 million versus less than one million in 1987. In Desert Storm, a specific mode within a given subsystem did not share data with other modes or, put another way, the data within a given function was separate. For example, the radar would lose the data from multiple air-to-air tracks generated in track-while-scan mode when commanded to switch to single-target-track.

Since Desert Storm, software updates have been developed and fielded to retain data from the previous mode when switching from one mode to another. The significance of exchanging data between subsystems was identified and software upgrades incorporated to establish connectivity between the various subsystems. For example, software was used to exchange information between the radar and the weapon control computer. By Operation Iraqi Freedom, we had taken a quantum leap forward sharing data between air, sea, and ground forces. Now one source can take a picture of the target, share that data with an F/A-18 loaded with the desired ordnance, and pass on the

coordinates and other information for rapid, precise targeting.

Today, the same core set of weapons used in Desert Storm is now capable of being redirected to a different target during flight. The efficiency and effectiveness of the weapons were increased through minor upgrades to software. Our ability expanded from delivering multiple weapons on a single target to delivering multiple weapons on multiple targets.

Navigation was dramatically improved with the introduction of GPS and digital moving maps in the F/A-18s. These additions improved situational awareness and sustained higher accuracy than the older inertial platform used in Desert Storm.

Operation Iraqi Freedom shifted the focus of combat. Our situational awareness expanded from focusing on a single F/A-18 mission and its intended targets to all forces (ground, air, weapon) coming together to achieve a single mission in a coordinated manner. Coordination at this level required rapid and accurate identification of forces.

The ability to quickly determine friendly from hostile contact was inherent to performing assigned tasks. Timely information exchange among the Navy, Air Force, Marines, Army, and coalition forces using common, easily understandable formats is the core element in our most recent deployment. First deployment of systems like the Digital Communication Set and Multifunctional Information Distribution System onboard the F/A-18 brought the battleforce commander the ability to flex the plan based on current information regarding target location, status, and lethality.

Future Growth

By 1991, it was becoming clear that avionics cooling, electrical, and space constraints would begin to limit future growth of the F/A-18 C/D. The multi-mission F/A-18 E/F *Super Hornet* strike fighter is an upgrade of the combat-proven F/A-18 C/D.

From an interoperable, total ownership cost viewpoint, the biggest advance is achievement of a 90 percent commonality of avionics between the C/D and E/F models. However, the F/A-18 E/F cockpit features a touch-sensitive, upfront control display; a larger, liquid-crystal multipurpose color display; and a new engine fuel display. The F/A-18 E/F aircraft are 4.2 feet longer than earlier Hornets, have a 25 percent larger wing area, and carry 33 percent more internal fuel that will effectively increase mission range by 41 percent and endurance by 50 percent.

The Super Hornet incorporates two additional weapon stations. This allows for increased payload flexibility by mixing and matching air-to-air and/or air-to-ground ordnance. The aircraft can carry the complete complement of *smart* weapons, including the newest joint weapons such as JDAM and JSOW.

Enabling Technologies and Processes

When the F/A-18 concept was first developed, software was an art rather than the science it is today. The engineering discipline has matured and expanded to include a systems view and commonly accepted process principles as contained in the Capability Maturity Model®. Predictable product delivery containing promised high-quality functionality within cost is the foundation of process improvement models.

"The reality is that software is the enabler that ties individual units of a battlegroup into a single striking entity."

The F/A-18 was the Navy's first tactical jet aircraft to incorporate a digital, multiplex bus architecture for the entire system's avionics suite. The benefit of this design feature is that the F/A-18 has been relatively easy to upgrade on a regular, affordable basis. The software architecture provides the basis for making more frequent updates to system capabilities [4].

Achieving an integrated system solution demands communication and coordination between the end user, requirements personnel, system and software designers, and test personnel, which has manifested itself in adoption of Integrated Product Teams to move from concept through development to product support. Today's expectations are for a flexible system solution that meets demands of any current combat situation while continuously formulating options for the future. Current expectations assume a solid foundation of individual software components working seamlessly together to provide needed capabilities, not unlike the way we expect our laptop to perform needed functions at any time, every time.

The advent of new high-order programming languages brings many benefits to the software development arena. The complexity of needs that can be addressed through software algorithms and processing is huge compared to just 10 years ago. The new Super Hornets utilize the more modular, object-oriented design features that did not exist when the original F/A-18s were rolling off the production line. Basically, what took the Navy 20 years to create as functionality for the F/A-18 aircraft was converted to the more cost-effective High Order Language (HOL) in five years with every warfighting function verified in two years. This recoding of functionality involved some 1.3 million lines of code. The effort involved delivery of more than 100 major warfighting capabilities (e.g., HUD, backup mode), containing well over 1,000 possible operator selections. The F/A-18 Advanced Weapons Lab was recognized with the 2003 U.S. Government's Top 5 Quality Software Projects award for the HOL conversion.

Commercial off-the-shelf (COTS) products are major enablers in converting to HOL. Our COTS-based system is the enabler for future capability enhancements to the F/A-18 production line. It enables the F/A-18 platform to grow and adds more computing horsepower on demand, for example, to expand the F/A-18's use from its current fighter/attack role into an electronic attack (EA) role currently provided by the Navy's EA-6B aircraft. The combination of COTS and HOL has made updating the aircraft's entire functionality more modular, economical, and faster.

The tools available to develop software have undertaken unimaginable leaps to support integrated teaming across geographically diverse locations. Even the tools used to generate software code have become more sophisticated and visual, making the effort required to design and perform low-level testing more cost efficient. The F/A-18 program uses commercial tool suites for software development. Examples include the desktop environment that allows developer testing to occur on a workstation versus a separate test facility. Another innovation was an automatic display code generator that shows promising use in flight simulations, test facilities, trainers, and technical publications.

Long before Operation Iraqi Freedom, the tide had turned from just looking at what software processing could be achieved within a single mode, a single box, or even a single F/A-18. The battlegroup is turning into a single-striking unit making it difficult to look back and focus on highlighting the software aspect of this amazing

® The Capability Maturity Model is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

evolution. We now realize our perception in the 1980s was that software was the end of the journey. The reality is that software is the enabler that ties individual units of a battlegroup into a single striking entity.

The F/A-18 program understands – and eagerly steps forward to engage in – the possibilities of net-centric warfare. The full potential of force multiplication relying on software is still to come; our most recent combat test with F/A-18 confirms the validity of our mission and software strategy.◆

References

1. Federation of American Scientists Military Analysis Network. "F/A-18 Hornet." Washington, D.C.: FAS, 2 Apr. 2004 <www.fas.org/man/dod-101/sys/ac/f-18.htm>.
2. Weaver, Susan. F/A-18 AWL Management and Systems Engineering Process Manual. F/A-18 Advanced Weapons Laboratory, June 2000.
3. Boeing. "Hornet: 20th Anniversary of First Flight." Boeing, St. Louis, MO; 2 Apr. 2004 <www.boeing.com/defense-space/military/fa18/fal820/stories.htm>.
4. GlobalSecurity.org. "F/A-18 Hornet." Alexandria, VA: GlobalSecurity.org, 2 Apr. 2004 <www.globalsecurity.org/military/systems/aircraft/f-18.htm>.

About the Author

Susan Weaver, a senior management application specialist with L-3 Communications Government Services, Inc., is the F/A-18 Advanced Weapons Laboratory's internal consultant on process improvement. She guides the organization in defining and documenting its processes as the organization evolves, and leads its process improvement program and chairs the Systems Engineering Process Group. Her experience spans software/systems engineering, weapons and tactics simulation analysis, and technical writing. Weaver has a Bachelor of Science in business project management.

L-3 Government Services, Inc.
330 East Ridgecrest BLVD STE B
Ridgecrest, CA 93555-5814
Phone: (760) 939-5793
Fax: (760) 939-5961
E-mail: susan.weaver@navy.mil

Web Sites

Object Management Group

www.omg.org

The Object Management Group (OMG) is an open membership, not-for-profit consortium that produces and maintains computer industry specifications for interoperable enterprise applications. Membership includes virtually every large company in the computer industry, and hundreds of smaller ones. Most of the companies that shape enterprise and Internet computing today are represented on the OMG board of directors. The OMG flagship specification is the multiplatform Model Driven Architecture (MDA), recently under way but already well known in the industry. The Object Management Architecture defines standard services that will carry over into MDA work. The OMG task force standardizes domain facilities in industries such as health care, manufacturing, telecommunications, and more.

Global Information Grid Enterprise Services

http://ges.dod.mil

The Global Information Grid Enterprise Services (GIG ES) is a suite of value-added information, Web, and computing capabilities that will improve user access to mission-critical data. GIG ES will provide access – anytime, anywhere – to reliable, decision-quality information through the use of cutting-edge, Web-based, networked services. GIG ES improves access to reliable decision-quality information for Department of Defense components at all echelons. End users can pull mission-tailored information intelligently from anywhere within the network environment with minimal latency. This will enable leveraging of best-of-breed concepts (many of them Web-based) and will maximize the net-centric performance of the GIG ES.

Defense Information Systems Agency

www.disa.mil

The Defense Information Systems Agency (DISA) is a combat support agency that is responsible for planning, engineering, acquiring, fielding, and supporting global net-centric solutions and operating the Defense Information System Network to serve the needs of the president, vice president, the secretary of defense, and the other Department of Defense components, under all conditions of peace and war. DISA's core mission areas include communications, combat support computing, information assurance, joint command and control, and joint interoperability support.

DACS and DSC Web Site

www.dacs.dtic.mil

The Defense Acquisition Contract Service (DACS) and Defense Software Collaborators (DSC) Web site aids in discovering Department of Defense (DoD)-sponsored software resources for program managers and software developers. The DACS has been designated as the DoD Software Information Clearinghouse serving as an authoritative source for state-of-the-art software information offering technical services designed to support the development, testing, validation, and transitioning of software engineering technology. The DACS has created the DACS Gold Practice Web site at <www.GoldPractices.com> to provide information about many prevalent software acquisition and development best practices that may have a positive impact on program risks and return on investment.

Federation of American Scientists

www.fas.org

The Federation of American Scientists (FAS) conducts analysis and advocacy on science, technology, and public policy, including national security, nuclear weapons, arms sales, biological hazards, secrecy, education technology, information technology, energy, and the environment. The FAS is a privately funded nonprofit policy organization whose Board of Sponsors includes 58 of America's Nobel laureates in the sciences.

American Institute of Aeronautics and Astronautics

www.aiaa.org

Today, with more than 31,000 members, the American Institute of Aeronautics and Astronautics is the world's largest professional society devoted to the progress of engineering and science in aviation, space, and defense. The Institute continues to be the principal voice, information resource, and publisher for aerospace engineers, scientists, managers, policy makers, students and educators.

Tomahawk Cruise Missile Control: Providing the Right Tools to the Warfighter

Marcus Urioste
Lockheed Martin Integrated Systems & Solutions

With fast-changing targets, unconventional enemies, and shadowy, pop-up targets of opportunity, our warfighters require the very best software solutions that take advantage of newest-generation cruise missile capabilities. The Tactical Tomahawk Weapon Control System gives the United States' and the United Kingdom's naval warfighters the right tools to carry out today's demanding strike missions.

The evening news program cuts to a videotape of a lone warship operating off a coastline far from home … the night sky is pierced by the brilliant flash of a cruise missile emerging from the warship's flush-mounted deck launcher, climbing, banking, and quickly disappearing over the horizon. A few miles away, the seascape is altered by another cruise missile emerging from the depths, sent on its way from a stealthy nuclear-powered submarine lurking beneath the waves. The attack is on, and Tomahawk cruise missiles are the first punch in the opening salvo.

Recent world events show that the United States and its coalition partners are being called upon to use *smart* weapons in both the prosecution of conflicts with other nations, and increasingly, in the global war on terrorism. Smart weapons in general and cruise missiles in particular are often the first surgical instruments of military power projection, focusing destruction only where intended while limiting the danger to our warfighters.

Improving the Tools

The Tactical Tomahawk Weapon Control System (TTWCS) is the next-generation system for planning and controlling Tomahawk cruise missile flight. The TTWCS development is part of the U.S. Navy's Tactical Tomahawk Weapon System Upgrade to improve the flexibility and responsiveness of Tomahawk cruise missiles, add new capabilities, and upgrade existing fleet systems.

The TTWCS' efforts include the full array of system development, including requirements definition, system engineering, system architecture and design, software development, software integration, hardware engineering, hardware manufacturing, hardware and software integration, system testing, logistics, training, and system installation. The TTWCS program will support U.S. surface ships and fast-attack submarines, and is planned for newly converted U.S. guided missile submarines and U.K. fast-attack submarines.

Tools in the Warfighter's Hands

The TTWCS was formally approved for initial operating capability in December 2003 to work with existing Tomahawk missiles in the nation's inventory. The TTWCS initial operating capability for the newest Block IV Tactical Tomahawk missile was achieved in mid-2004. The U.S. Navy began fleet installation of the TTWCS system in early 2004 and could provide up to one hundred new weapon control systems by 2008.

> *" …TTWCS adds much more capability to control the Tomahawk missile(s), direct, redirect, mission plan, and replan, while at the same time keeping the system interfaces easy to use for the officers and sailors onboard."*

The Right Team

The TTWCS program consists of a multi-disciplinary team (systems engineers, software developers, system testers, hardware engineers, and logistics and training specialists) composed of the acquisition agent, Naval Air Systems Command for Cruise Missile Weapon Control Systems, Patuxent River, MD; the Naval Surface Warfare Center Division, Dahlgren, VA; the Naval Undersea Warfare Center Division, Newport, R.I.; and the prime contractor, Lockheed Martin Tactical Control Systems, Valley Forge, PA.

Tomahawk Command and Control Legacy

The U.S. Navy's Cruise Missile Program has been effectively evolving for almost 30 years. The original Tomahawk Weapon Control System effort started in the late 1970s, and the follow-on Advanced Tomahawk Weapon Control System (ATWCS) program lasted from June 1993 to December 1998.

The ATWCS was a large-scale hardware and software integration and software development program to replace the Tomahawk cruise missile shipboard operational hardware and software system. The ATWCS team designed, developed, and integrated software products from commercial corporations and U.S. Navy laboratories into a cohesive, multi-security level weapons control system; conducted system level tests of the integrated products; prototyped future requirements; and successfully implemented the ATWCS on U.S. Navy surface ships and submarines.

Lockheed Martin won a competitive program to provide the third-generation cruise missile weapon control system, the TTWCS, in May 1999.

Benefits to the Warfighter

Shortening the Timeline

The TTWCS system will reduce the warfighter's Tomahawk timeline by bringing the missile mission planning function aboard the firing unit. Previously, this planning function was done solely at shore-based or dedicated afloat cruise missile support centers, taking much more time to replan and provide new or revised missions to the ship.

The system's Launch Platform Mission Planning component is a new capability that reduces weapon system reaction times by speeding up the tactical mission planning process. Another new capability is the ability to redirect missiles to new targets while in flight, available with the newest Block IV Tomahawk. These newest capabilities are particularly important today, with fast-changing targets,

unconventional enemies, and shadowy, pop-up targets of opportunity.

Ease

The new TTWCS program took on the task of providing the next-generation cruise missile weapon control system to handle the newest technological improvements to the Tomahawk missile, while keeping the system user-friendly enough to be maintained by young shipboard operators not far removed from high school graduation. Simply put, the TTWCS adds much more capability to control the Tomahawk missile(s), direct, redirect, mission plan, and replan, while at the same time keeping the system interfaces easy to use for the officers and sailors onboard.

Space

Space on ships and submarines is in great demand by equipment to power and protect the ship, by weapon systems, and by people. The TTWCS successfully reduced the command-and-control equipment footprint onboard from seven racks of computer equipment to three, all while adding vital new capabilities.

Software's Vital Role in the TTWCS

Multiple Platforms

The TTWCS is being installed or is planning to be installed on U.S. Navy surface ships (destroyers and cruisers), U.S. Navy converted guided missile submarines, and fast-attack submarines (Los Angeles class/Virginia class/U.K.'s Trafalgar and Astute class). On U.S. fast-attack submarines, the TTWCS runs on the Combat Control System common hardware. The TTWCS surface-ship environment consists of multiple, redundant, single-board computers, running UNIX and Windows operating systems. The fully redundant, VME-based hardware architecture is housed in two TTWCS equipment cabinets. The software is executed by operators at four tactical display consoles on surface ships and from one to four consoles on submarines. The TTWCS interfaces with several shipboard systems, including the inertial navigation system, weapon vertical launch system, the global command-and-control system – maritime, and communications networks.

The Right Software

The TTWCS is a major software-based reengineering upgrade to implement even greater warfighter capability over previous Tomahawk missile control generations. The program's software development is certified at the Software Engineering Institute's Capability Maturity Model® (CMM®) Integration (CMMI®) Level 5. The software development incorporates a variety of components that spans new development, reused software from the predecessor ATWCS program, and government and commercial products.

The TTWCS software consists of six computer-software configuration items with approximately 500,000 lines of new and modified development code and 500,000 lines of reused code. The delivered software is C, C++, Java, and Ada, and is developed to be compatible with the Defense Information Infrastructure Common Operating Environment (DII COE). The DII COE is a Department of Defense-wide common operating environment that enables common standards and implementation tools for military tactical situational awareness and system interoperability. Prior to DII COE, each new military system requiring situational awareness and interoperability developed individual solutions.

"The TTWCS benefits from an incremental software development model that offers improved quality, reduced cost, and better adherence to schedule over spiral or waterfall development."

The TTWCS program reached a DII COE Level 7 (8 is highest possible), signifying virtually no duplication of DII COE functions within the system application. The TTWCS demonstrated the benefits of the DII COE reuse concept through reduction in development and life-cycle costs. The DII COE software is incorporated into the TTWCS infrastructure layer, which minimizes redundant code and maximizes consistency for system services and evolution to newer computing platforms. Software development is accomplished using object-oriented methodologies and Common Object Request Broker Architecture for interfaces among software components.

The Right Development Environment

The TTWCS software development environment consisted of a network of Hewlett Packard (HP) workstations (B-180Ls and C-110s) and servers (K-360 and K-580 mid-class), all running HP UNIX. Engineers used desktop PCs to access the development network via XOnNet. Tools used include Popkins' System Architect (system/software architecture design); Telelogic's COOL:Jex (detailed software design); Telelogic's DOORS (requirements traceability); IBM's ClearCase (configuration management); IBM's ClearQuest (problem reporting/resolution database); HP's SoftBench (C/C++ compiling and debug); ADA Core Technology's GNAT (Ada compiler and coding); and Aonix' Teleuse and Builder Xcessory (human-computer interface display generation).

The Right Development Model

The TTWCS benefits from an incremental software development model that offers improved quality, reduced cost, and better adherence to schedule over spiral or waterfall development. Each increment adds functionality and is taken through a full development and test cycle. In software increment one, legacy software from ATWCS was integrated with TTWCS hardware. During increment two, the TTWCS infrastructure (Operating Environment and Common Services Middleware Layer) was implemented and matured. Software increments three through six added new functionality to the heritage weapon control system. The final increment contained the full TTWCS functionality and was formally tested under rigorous supervision.

Incremental software development allows for risk reduction through incremental system integration. Each successive increment is developed at reduced technical risk due to a solid functional and performance foundation established in the preceding increment.

Software Metrics

The TTWCS contract was awarded to Lockheed Martin in May 1999. The U.S. Navy and Lockheed Martin jointly determined that the TTWCS would be a focus program for implementing CMM Level 5 processes and supporting tools. In

® CMM and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

December 2000, the TTWCS was scored at CMM for Software Level 5. In June 2002, Lockheed Martin Management & Data Systems (now Lockheed Martin Integrated Systems & Solutions) became one of the first companies in the world to achieve CMMI Level 5.

The implementation of Level 5 processes on the TTWCS has resulted in productivity improvements, defect reductions, and cost savings for the U.S. government. CMMI Level 5 has provided measurable improvements in software development: a 30 percent increase in software productivity, a 20 percent drop in development costs, and a 15 percent drop in detect/find/fix software cost.

- **Productivity.** Based on historical metrics for similar developments, using CMM Level 5 processes resulted in a reduction of over 15,000 development hours.
- **Quality.** In-process quality activities enabled early problem detection during design and code/unit test, resulting in a 30 percent reduction in defects during integration and verification.
- **Integrated Program Environment (IPE) and Integrated Development Environment (IDE).** The IPE enabled everyone on the program, regardless of location, to participate in daily decision making. The IDE enabled collaborative development among 500+ users across the United States.
- **DII COE Compliance.** The delivered software achieves DII COE Level 7 compliance for newly developed software and Level 5 for reused software. The DII COE software is incorporated into the TTWCS infrastructure layer, which minimizes redundant code and maximizes consistency for system services.
- **Global Command-and-Control System – Maritime Interoperability.** The Weapon Control System (WCS) Common Services minimizes redundant code and maximizes consistency for all system services by using services developed and maintained by the U.S. Navy.

Getting the Process Right

The TTWCS benefits from high quality, defect prevention, improved productivity, and reduced risk inherent in achieving the industry's highest level of software process maturity. The program also complies with ISO 9001 objectives. The TTWCS Product Assurance Plan spans the entire program life cycle and ensures adherence to mandatory processes; development of compliant software, hardware, and documentation; and application of quantitative management (metrics) techniques. The plan applies to all locations where program activities occur. The plan's proactive product assurance methodology encompasses the following:

- Preventive action and continuous process improvement.
- In-process inspections and process and product compliance audits at all sites.
- Root-cause analysis to identify areas for improvement, increased product quality, and reduced risk.
- Metric collection and analysis to identify areas for process improvement early in development and throughout the entire life cycle.

The Right Test Environment

Team Approach

The TTWCS test-approach leveraged facilities and personnel located at government and contractor facilities. This approach allowed the team to evaluate and test the Tomahawk Weapon System from multiple perspectives, ensuring a robust test program that reduced redundancy and validated the system's capability to meet the warfighter's needs. Finding problems early and getting timely fleet feedback in the development cycle reduced follow-on development costs.

At-Sea Testing

To support at-sea testing of the weapon control system as part of the larger weapon system, the TTWCS team validated the performance of all software builds produced by the WCS software development team prior to the installation aboard the test ship, U.S.S. Stethem, an Arleigh Burke class-guided missile destroyer. The team established a configuration that permitted the U.S.S. Stethem to simulate communication with multiple Tactical Tomahawk missiles, thus enabling the completion of the technical evaluation shipboard event.

The configuration allowed testing of high numbers of missile launches and in-flight communications, utilizing non-expended assets that offer repeatability, sustainability, and high accuracy at an established one-time cost for procurement and low operational cost. The test ship was able to perform missile redirection of multiple missiles simultaneously and request health and status information from the missiles during flight. The missile simulation responded with both scheduled and unscheduled health and status event messages as directed by the communications plan, resulting in a successful test event.

Summary: Providing What Matters

The TTWCS program's success to date in providing the warfighter with the very best Tomahawk cruise missile control capabilities is a direct result of the dedicated team that stands behind this vital warfighter system. Through a combination of strong warfighter guidance and contractor performance, adoption of industry best practices, and the exercise of innovative technical solutions, the TTWCS team has provided even more timely capabilities for the warfighter. Ships and submarines will have cruise missile capabilities that exceed even the successful capabilities seen recently in Kosovo, Afghanistan, and Iraq. In a hostile world where a surgical strike can be needed in a moment's notice, the TTWCS team has provided the capability that gives our warfighters exceptional flexibility, versatility, and timeliness to address threats to our nation.◆

About the Author

Marcus Urioste leads business development and internal research and development programs for Lockheed Martin's Tactical Control Systems in Valley Forge, PA. His recent experience includes business development in advanced technology, and in international business development for Lockheed Martin Global Telecommunications. He previously served with distinction as a nuclear-trained submarine officer in the U.S. Navy on two fast-attack submarines, and as a tactics instructor at Naval Submarine School. He has been published previously in the U.S. Naval Institute's "Proceedings." Urioste is a Phi Beta Kappa graduate in mathematics of Tulane University, where he received his Navy commission via the Naval Reserve Officers Training Corps program.

Lockheed Martin Integrated Systems & Solutions
230 Mall BLVD
BLDG 100/RM 1237
King of Prussia, PA 19406
Phone: (610) 354-3808
Fax: (610) 354-7216
E-mail: marcus.m.urioste@lmco.com

Service-Oriented Architecture and the C4ISR Framework

Dr. Yun-Tung Lau
Science Applications International Corporation

This article presents an architecture modeling approach for formulating service-oriented architectures such as those being developed on the global information grid. The approach uses object-oriented techniques to supplement the traditional Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance Framework.

The global information grid (GIG) is the globally interconnected, secured, end-to-end set of information capabilities, associated processes, and personnel for collecting, processing, storing, disseminating, and managing information on demand to warfighters, policy makers, and support personnel [1]. The GIG supports all U.S. Department of Defense (DoD), national security, and related intelligence community missions and functions. It provides capabilities from all operating locations and interfaces to coalition, allied, and non-DoD users and systems.

The GIG as a transformational vision aims at achieving information superiority in a network-centric environment. It enables various systems to interoperate with each other. For the warfighters, it brings *power to the edge* through a Task, Post, Process, Use process. For the business and intelligence communities, it provides the infrastructure for effective information gathering and collaborative operation.

This transformation from a centralized, sequential thinking and a static one-to-one interfacing paradigm to a distributed, parallel information sharing and dynamic collaboration approach requires a fundamental shift in the way systems are built. Specifically, it lends itself to a service-oriented architecture (SOA) on a ubiquitous network carrying information on demand.

In an SOA, a set of loosely coupled services works together seamlessly and securely over a network to provide functionalities to end users. These services have well-defined interface contracts. Supported by service management tools at the enterprise level, they are published, discovered, mediated, and consumed in an orderly fashion.

The service-oriented approach is inherently dynamic. It allows fast formation of expedient communities of interest (COI) to handle highly volatile situations and changing mission requirements. It also supports the stable operation of longstanding or institutional COIs. The SOAs are flexible because each service encapsulates the underlying platforms and technologies that support it. The services provided at the enterprise level are therefore agnostic to those specific platforms and technologies.

The Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance (C4ISR) Architecture Framework [2], along with its three standard views and common products, has been widely used in building C4ISR systems. The framework was originally developed in 1996 by the DoD to provide guidance for describing architectures. Version 2 was officially mandated in 1998 as the DoD Architectural Framework and is being superceded by the DoD Architecture Framework (DoDAF). Standard techniques employed by C4ISR include point-to-point interfacing, static connectivity, and data-flow analysis. These are more suited to the traditional sequential processing, system-oriented, and one-to-one integration paradigm.

"In an SOA [service-oriented architecture], a set of loosely coupled services works together seamlessly and securely over a network to provide functionalities to end users. These services have well-defined interface contracts."

With the ongoing efforts to transform the DoD into a network-centric, service-oriented environment, the following questions are often raised:

- Does the C4ISR framework apply to such a service-oriented architecture?
- How is it supplemented with other techniques to fully describe such architectures?
- What are the set of products that describe the essence of a service-oriented architecture?

Whereas full answers to these questions will require extensive discussion, this article describes a pragmatic approach that naturally fits the service-oriented environment. This approach utilizes object-oriented design and analysis techniques to supplement the standard C4ISR framework for developing SOAs. As the DoD moves toward a network-centric environment supported by SOAs, this approach provides a timely and rigorous methodology for specifying future enterprise architectures, and has been recently applied to the architecture development of Net-Centric Enterprise Services (NCES) with satisfactory results.

The NCES is a collaborative environment that supports vertical/horizontal interoperability between DoD business and warfighting domains, as well as the national intelligence domain [3]. The NCES provides the core enterprise services that support various standing and expedient COIs on the GIG.

Using this approach, the use cases for the NCES core enterprise services were developed. The corresponding operational and systems views were constructed as part of an integrated architecture product. Activities in the use cases were also mapped to the Net-Centric Operations and Warfare Reference Model [4].

In the following sections, I first describe the approach for formulating an SOA. Using an enterprise messaging system as an example, I then discuss the corresponding architecture views that embody the approach in the C4ISR framework[1].

Formulating an SOA

In formulating an SOA, you start with operation. Here the focus is how end

Architecture Views and Products

The Department of Defense (DoD) Architecture Framework (AF) (which will supercede the Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance Architecture Framework) provides the guiding principles for modeling and designing architectures in the DoD environment (Version One is available at <www.aitcnet.org/dodfw>). Architecture is described by three views:

- The Operational View (OV): Describes the tasks, activities, operational elements, and information exchanges required to accomplish missions.
- The Systems View (SV): Describes systems and interconnections supporting operational functions.
- The Technical View (TV): Includes technical standards, implementation conventions, rules, and criteria that guide systems implementation.

Each view has a set of products. Some are listed in the table below (the product numbers do not imply the order of developing them). In addition to those listed, there are 12 products and two *All Views* products omitted for brevity.

Views	Products and Descriptions
Operational	OV-1 (high-level operational concept graphic)
	OV-2 (operational node connectivity description)
	OV-3 (operational information exchange matrix)
	OV-5 (operational activities model)
	OV-6c (operational event-trace description)
Systems	SV-1 (systems interface description)
	SV-2 (systems communications description)
	SV-4 (systems functionality description)
	SV-5 (operational activity to systems function traceability matrix)
	SV-6 (system data exchange matrix)
	SV-11 (physical schema)
Technical	TV-1 (technical standards profile)

users, systems, or applications use services. Use cases in Unified Modeling Language (UML) [5] describe the external behavior of a service as seen or utilized by an actor (user, system, or application). The boundary of the service in a use case is clearly delineated. The interaction of the actor with the service is described without revealing the internal details of the service. Use case is, therefore, a natural tool for describing operational activities in an SOA.

Figure 1: *Formulating an SOA*

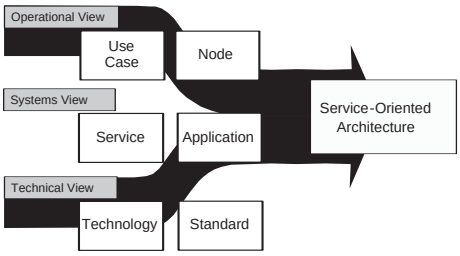
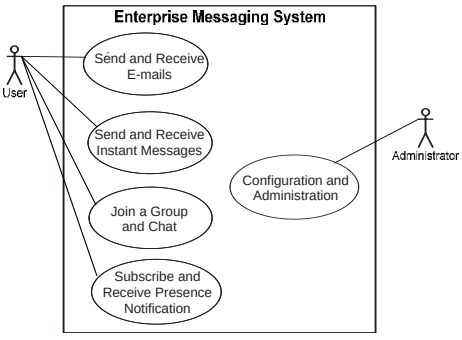


Figure 2: *Use Cases as Part of the Activity Model (OV-5) for the Enterprise Messaging System*



Based on the operational concept, the scope of services, and the high-level requirements, one may identify a set of high-level critical use cases. These are the use cases that the architecture must support to meet the minimal requirements. Use cases are not requirements. Nevertheless, they illustrate what functions architecture provides and highlight the requirements. Therefore, use cases are the first step in formulating an SOA (see Figure 1).

In each use case, typically two or more nodes interact with each other by exchanging information. If a node is a service consumer, then it is an actor in the use case for that service. If it is a provider, then it is a component providing that service. Traditionally, a node in the C4ISR framework represents a role, an organization, an operational facility, etc. For an SOA, its scope is expanded to include shared resources and services. Hence a service node interacts with consumer nodes to provide services. Use case and node are, therefore, the primary objects in describing the operational aspects of an SOA, as shown in Figure 1.

Once the operational aspects are identified, the next action is to find the solution that satisfies the operational requirements. In an SOA, each service provides a set of well-defined functions useful to its users, or consumers. An example is chat service, which allows users to perform online chat. A set of services may interact in an orderly manner to provide a complete set of mission functions. In this way they form a mission application, or simply application. Application is not just a simple collection of services, but an integral set of logically connected services. Application and service form the primary objects that describe the systems aspects of an SOA. They are, in practice, the software that needs to be built by developers.

Finally, standards and technologies are the primary objects that constitute the technical foundation in implementing an SOA. Figure 1 summarizes the approach for formulating an SOA, along with the primary objects. Discussed next are the corresponding architecture views that embody the approach in the C4ISR framework[1].

Operational View

For a concrete example, let us consider an enterprise messaging system, which encompasses e-mail, instant messaging (IM), chat, and presence services. The critical use cases are send and receive e-mails and instant messages, participate in a chat session, subscribe to and receive presence notifications, etc. They are shown in the use case diagram in Figure 2. In addition, the administrator configures and administers the services.

For each use case, you may describe a sequence of events or activities. These activities may be presented in a hierarchy, as in the standard activity model operational view (OV-5). Here, however, the use cases provide a natural grouping of those related activities. Additionally, a use case highlights the actors and system/service boundary, allowing you to delineate roles and nodes easily. Hence, include use case as part of OV-5 and consider it an essential product for an SOA.

For an SOA, the use-case diagrams (such as Figure 2) often identify the nodes. These nodes are roles, organizations, shared resources, or service nodes. You can further draw the connections (i.e., the need lines) between the nodes, thereby forming the operational node connectivity description (OV-2). An example is given in Figure 3. Except for the use of UML deployment diagram

notations, this figure is the same as the standard OV-2.

Finally, a description of each connection in Figure 3 gives the operational information exchange matrix (OV-3). Each row in the matrix describes the provider and consumer nodes, the information exchange, the mode of exchange (e.g., synchronous), the security aspect, etc. This again is the same as the standard OV-3.

The high-level operational concept graphics (OV-1) still applies to an SOA. This, together with OV-5, OV-2, and OV-3, encompasses the concepts of operation, the use cases from user's viewpoint, the connectivity between operational nodes, and their information exchanges. They therefore characterize the essential operational aspects of an SOA. Furthermore, since operational nodes include shared resources and services, dynamic and collaborative operational activities are properly captured.

To analyze more details in the use cases, you may use the Integration Definition for Function Modeling process diagrams [6] to depict the activities (including inputs, controls, outputs, and mechanisms). This will also be part of OV-5. Alternatively, you can use the UML sequence diagrams (OV-6c) to describe the details.

Systems View

As discussed earlier, application and service are the primary objects in Systems View (SV). In an SOA, one is more concerned with the logical interaction between service providers and consumers. Rather than relying on static connections, users in an SOA may select different services under different use cases. Consequently, system interface description (SV-1) in the form of logical architecture diagrams is usually appropriate. Logical architecture diagrams show the connectivity between service provider nodes (or components) and consumer nodes. They also specify the types of interface or communication protocols.

For efficient software management, closely related use cases utilizing similar services are usually grouped together and supported by an application. Thus you may draw a functional decomposition diagram as systems functionality description (SV-4). The diagram will identify the applications. Each application supports one or more use cases and may have a corresponding logical architecture diagram as SV-1.

Going back to the sample enterprise messaging system, the basic services are e-mail, IM, chat, and presence services. E-mail service is a familiar form of asynchronous messaging and may be provided through either a thick or thin client. The other three services emphasize synchronous interaction and therefore are best provided through a single application to the users. The corresponding systems functionality description (SV-4) is shown in Figure 4.

The SV-1 picture for the e-mail service is shown in Figure 5. Here again, notations similar to the UML deployment diagrams are used. The boxes represent nodes that are connected by channels of data exchange. A service node has a well-defined service interface (indicated by a protruded match head) and supports data exchange in certain protocols. In Figure 5, the protocols and interfaces are HyperText Markup Language (HTML), HyperText Transfer Protocol (Secured) (HTTPS), Simple Mail Transfer Protocol (SMTP), Internet Message Access Protocol (IMAP), and the mail access application programming interface called Post Office Protocol v.3 (POP3).

Services are often organized into layers, with the lowest layer containing core services, and the upper layers containing value-added and composite services. Services in the upper layers use those in the lower ones to perform specific functions. You may use service layer diagrams such as SV-1 to show the dependencies of such service stacks. An example for the enterprise messaging system is shown in Figure 6 (see page 14), which includes the synchronous messaging services and storage and security core services. Note that a service consumer (such as IM and Chat Client) may dynamically connect to one of many chat servers that provide chat service. In this sense, the connectivity is not static.

There are situations such as in security service or network management service in which a system communications description (SV-2) is more suitable than SV-1. This is because such services are naturally associated with physical systems and network elements.

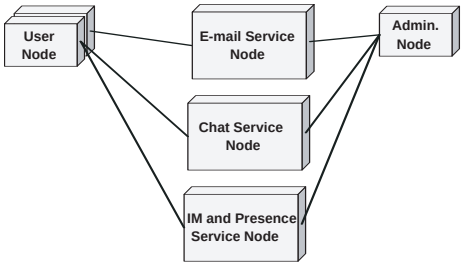


Figure 3: *Operational Node Connectivity Description (OV-2) for the Enterprise Messaging System*

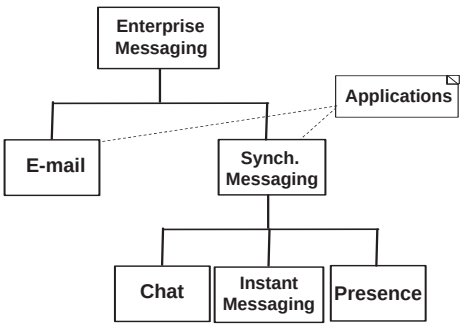
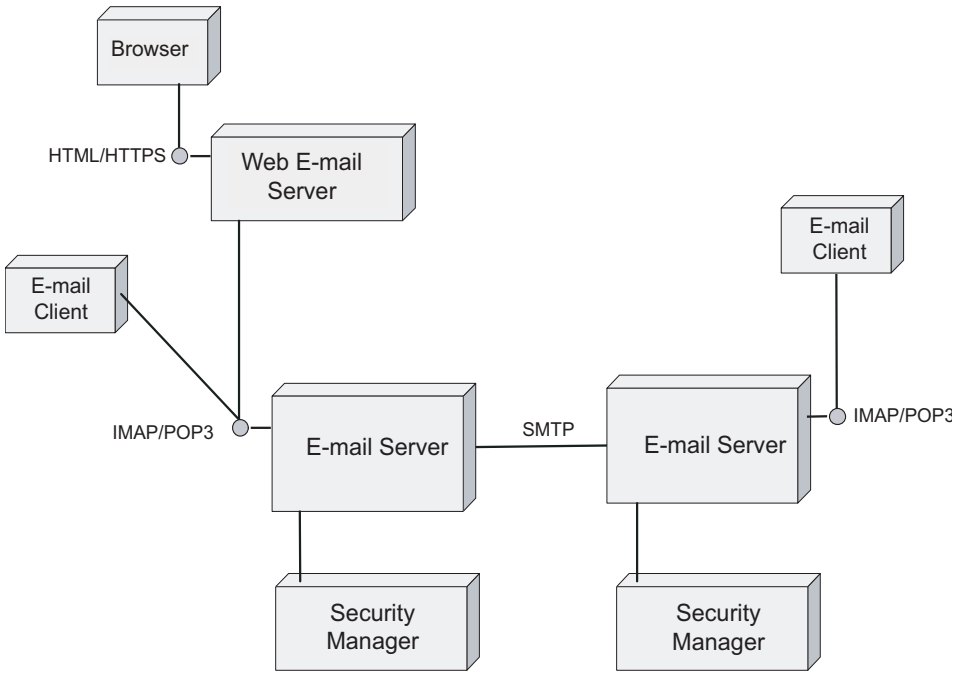


Figure 4: *Systems Functionality Description (SV-4) for the Enterprise Messaging System*

Figure 5: *Logical Architecture Diagram (SV-1) for the E-mail Service in the Enterprise Messaging System*



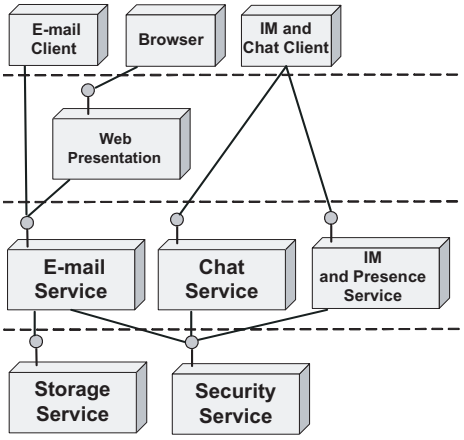


Figure 6: *The Enterprise Messaging System Represented as a Service Stack (SV-1)*

For an SOA, SV-4 and SV-1 (or SV-2), capture the functional breakdown and the logical or physical structures that support those functions.

Traditionally, system data exchange matrix (SV-6) provides detailed data exchange information between system nodes. Such a matrix depicts static data exchange connections. In contrast, data exchange in an SOA is specified by service contracts and a list of consumers of the services. Services can be dynamically published through a service broker. Consumers may then dynamically discover, subscribe, and consume the services. Hence service contracts play the role of SV-6 and the service broker facilitates connection between consumers and providers.

For instance, the emerging Web service paradigm uses Web Services Description Language to describe service contracts. Simple Object Access Protocol is used as the transport mechanism. And Universal Description, Discovery, and Integration may be implemented as part of the service broker.

In addition to SV-6, other SVs may also capture other supplementary properties of a service. For example, physical schemas (SV-11) may be used to describe data schemas in a service contract, and system performance parameters for quality of service or service level agreement. The operational activity to systems function traceability matrix (SV-5), on the other hand, shows how the applications satisfy the requirement by supporting the use cases.

Technical View

The Technical View (TV) in an SOA is the same as traditional C4ISR architectures, with standards and technologies as key elements. Here technical standards profile (TV-1) is essential because it references the key technical standards and technologies employed by the SOA. The Joint Technical Architecture [7] provides primary references for these standards and technologies.

Summary

Using UML techniques to supplement the traditional C4ISR framework, I have elucidated an approach for formulating an SOA. On the operational side, it starts with use cases, which involve the interaction of two or more operational or service nodes. Mission functions are provided through applications, which are implemented by a set of services. The corresponding C4ISR architecture products are also discussed along with an example.

In the appendices[1], I also present the complete UML model for architectural products in an SOA and its mapping to the Federal Enterprise AF. It provides a solid modeling foundation for the above approach.

When applied to NCES, this approach was very effective. The use cases in the nine core enterprise services of NCES drove the development of the architectural products. They also provided a natural link to the NCES requirements and direct connection to the end users. After developing the high-level architecture products, detailed events for the use cases were analyzed, service interfaces or contracts were defined, and metrics for service performance were established.

Some topics for future investigation on this approach include how to capture SOA products in a Core Architecture Data Model database, which is included in the DoDAF; how to specify and manage service contracts in an SOA to ensure interoperability across the enterprise; and how to evaluate compatibility or compliance between different SOAs.◆

References

1. U.S. Joint Forces Command. Global Information Grid Capstone Requirements Document. JROCM 134-01. Norfolk, VA: USJFCOM, Aug. 2001 <https://jdl.jwfc.jfcom.mil>.
2. C4ISR Architecture Working Group. C4ISR Architecture Framework Vers. 2.0. Washington, D.C.: Department of Defense, 18 Dec. 1997 <www.fas.org/irp/program/core/fw.pdf>. The next version is the DoD Architecture Framework Vers. 1.0. 15 Aug. 2003 <www. aitcnet.org/dodfw>.
3. Global Information Grid Enterprise Services. Initial Capabilities Document for Global Information Grid Enterprise Services. Arlington, VA: GIG ES, 9 Sept. 2003 <http://ges.dod.mil>.
4. Net-Centric Operations and Warfare Reference Model, Draft Vers. 1.0. 20 Oct. 2003 <https://disain.disa.mil/ncow. html>.
5. Object Management Group, Inc. Unified Modeling Language (UML), Vers. 1.5. Needham, MA: OMG, Mar. 2003 <www.omg.org/technology/documents/formal/uml.htm>.
6. National Institute of Standards and Technology. Integration Definition for Function Modeling (IDEF0). Federal Information Processing Standards Publication 183. Gaithersburg, MD: NIST, Dec. 1993.
7. JTA Development Group. Joint Technical Architecture Vers. 5.1. Washington, D.C.: U.S. Department of Defense, 12 Sept. 2003 <www.jtaonline.disa.mil>.

Notes

1. Additional details on this and other developments can be found in this article's online version at <www.stsc.hill.af.mil/crosstalk>. In the PDF version, click on the appendices link.

About the Author

Yun-Tung Lau, Ph.D. is assistant vice president of Technology at Science Applications International Corporation. He has been involved in large-scale software architecture, design, and development for 14 years. Lau has served as chief architect for many software and enterprise architecture projects, from scientific computing, to electronic commerce, to command and control systems. He has published many articles in professional journals and has written "The Art of Objects: Object-Oriented Design and Architecture." Originally trained as a theoretical physicist at Massachusetts Institute of Technology, Lau also has a Master of Technology Management.

Science Applications International Corporation
5107 Leesburg Pike STE 2000
McLean, VA 22041
Phone: (703) 824-5817
Fax: (703) 824-5836
E-mail: yun-tung.lau@saic.com

Executable Specifications: Language and Applications

Dr. Doron Drusinsky and Dr. J. L. Fobes
Naval Postgraduate School

With recent industry focus on software safety and dependability, and with a particular Department of Defense (DoD) emphasis on safety-critical systems, formal methods are gaining renewed attention as a way to ensure that an implementation is consistent with its specifications. Unlike conventional testing methods, which are human intensive and therefore slow, expensive, and error-prone, formal methods enable automated computer-aided verification. This article describes an easy-to-use formal method based on executable specifications. In particular, it describes the logical foundation of executable formal specifications, along with some interesting applications relevant to DoD missions such as run-time monitoring of real-time systems and online temporal rule checking for security-based applications. It also describes two categories of techniques for executing formal specifications.

Formal specification languages have been thoroughly investigated during the past three decades. They have been considered primarily for verification and validation purposes using techniques commonly known as formal methods. Most formal methods have suffered from limited commercial success due to several limiting factors such as their prohibitive computational complexity and the high level of mathematical skills needed to be used effectively.

Recent research has focused on *executable specifications*, a new class of applications of formal specifications whereby specification rules are executed on a computer much like any high-level programming language. This class of techniques and associated tools harnesses the linguistic power of formal specification languages yet is simple and does not suffer from the complexity limitations of formal verification methods. In addition, executable specifications enable new application domains in addition to classical verification such as online temporal reasoning in security applications.

In this article, we focus on temporal logic that is a particular and prominent formal specification language. We begin with background information about logic and formal methods and then describe temporal logic in greater detail. Next, we describe executable specification methods and tools followed by a description of a successful verification effort using executable specification. Lastly, we describe security rule-checking using executable specifications.

Background

Formal specification languages are designed to capture requirements (what a system should do) in a formal way, i.e., using mathematics. In contrast, design and programming languages capture the implementation (*how* a system implementation does what it is supposed to do).

Using mathematical notation to capture specifications removes potential ambiguity and, when coupled with mathematical proof techniques, enables program correctness proofs. These proofs provide indisputable statements about the absolute absence of errors in the implementation. This contrasts with testing techniques where only incomplete evidence is provided. The body of knowledge involving formal specifications and formal correctness techniques is commonly referred to as *formal methods*.

> *"Using mathematical notation to capture specifications removes potential ambiguity and, when coupled with mathematical proof techniques, enables program correctness proofs."*

Clearly, there is an inherent trade-off between investing in the education and tools of formal methods versus the potential benefit of assuring bug-free software. For example, a low-end Web site owner might be willing to take the risk of having program errors on the site rather than invest in costly verification methods. On the other hand, the cost of a single bug in the software onboard a multimillion dollar space mission justifies the investment in robust verification techniques such as formal methods.

The most popular mathematical domain used by formal specification languages is logic. In its simplest form, Boolean *propositional logic*[1] is the kind of logic found in every modern programming language such as the C/Java expression (x>0) && (y==1). However, propositional logic is not powerful enough to elegantly capture temporal and aggregational aspects of the system. For example, propositional logic cannot explicitly state that (x>0) *must be true now* and (y==1) *must be true sometime within the next 5 seconds*.

First Order Logic (FOL) extends propositional logic with two quantifiers: the universal quantifier (∀ read as *for all*), and the existential quantifier (∃ read as *there exists*). These quantifiers range over a known set, i.e., the set of all cars registered in California. Hence, a statement such as *a California registered minivan must be at most 10 feet long* can be stated in a single expression: ∀car: minivan → (length≤10ft.).

In contrast, using propositional logic would require that you explicitly state – for every car in the set – the above statement. Also, using programming techniques to achieve the desired aggregate effect defeats the whole purpose of specification, i.e., to make a clear statement about *what* the system should do without dealing with the *how* it does so.

Linear-Time Temporal Logic (LTL), the formal specification language described later in this article in the section "REM Tools: Code Generators and Monitors," extends propositional logic with four temporal operators. LTL has an advantage over FOL in that it removes mathematical clutter and enables specifications in a form that is close to natural language. It is mostly suitable for *reactive systems*, i.e., systems that constantly interact with their environment such as control software in a cruise missile.

Two primary classes of formal correctness proof techniques are theorem

provers and model checkers. Theorem provers use logic proof methods to prove that a program conforms to a given specification. Theorem provers support only a subset of LTL, and typically require a highly skilled human driver. Model checkers, on the other hand, are automatic and support full-blown LTL specifications (though typically with little support for real-time constraint validation). However, due to their prohibitive computational complexity, model checkers tend to work well only for small programs.

Run-time Execution and Monitoring (REM) is an effective and efficient hybrid between formal methods and conventional execution or simulation-based testing techniques. REM uses LTL-based specifications augmented with real-time constraint specifications. REM is a method of automatically comparing the behavior of an underlying application such as an embedded system to its formal specification. This is done by executing the specification in tandem with the application.

While REM uses formal specifications, it is not a pure mathematical proof technique; test-based verification and corresponding test suites are still required. Nevertheless, REM is simple to use and automates the verification process. In addition, REM tools described in the sequel are capable of detecting real-time requirement violations while executing on an embedded target. Interestingly, REM is also useful for non-verification applications such as security checking, as described in the "Security Applications" section of this article.

Formalism and Language Lessons

LTL is an extension of propositional logic that deals with time and order. As early as 1977, LTL was proposed as a way to formally specify multithreaded programs [1, 2]. Since then – and especially during the last decade – researchers have expanded its theoretical and practical power using it, for example, to specify protocols and hardware. LTL is a simple and intuitive extension of propositional logic that is closer to natural language than most other specification languages. For those reasons, LTL is the formal specification language used by most formal methods and is also the specification language of choice for REM methods and tools.

The syntax of LTL adds eight operators to the AND, OR, IMPLIES, XOR, and NOT of propositional logic. Four of the operators deal with the future: Always in the future, Eventually (sometime in the future), Until, and Next cycle; additionally, four dual operators address the past: Always in the past, Sometime in the past, Since, and Previous cycle.

Metric Temporal Logic (MTL) enhances LTL's capabilities [3]. With it, you can define upper and lower time constraints as well as time ranges for the LTL operators. By imposing relative and real-time constraints on LTL statements, MTL lets you use LTL to verify real-time systems. The following is an example showing a relative-time upper boundary in MTL:

$$\text{Always}_{<10}(\text{readySignal Implies Next ackSignal})$$

which reads,

> Always, within the next 10 cycles, readySignal equals 1 implies that one cycle later ackSignal equals 1.

The text inside the parentheses is a propositional logic expression, and a cycle is an LTL time unit, which is a user-controlled quantity. The time constraint is relative in that it is counted in clock cycles.

Another example is as follows:

$$\text{Always}_{\text{timer1}[5,10]}(\text{readySignal Implies Eventually}_{\text{timer2}>=20}\ \text{ackSignal})$$

which reads,

> Always, between 5 and 10 timer1 real-time units in the future, readySignal equals 1 implies that eventually, at least 20 timer2 real-time units further in the future, ackSignal equals 1.

Here, two real-time constraints are specified using timer1 and timer2 clocks. A separate statement maps these timers to system calls, system clocks, or another counting device.

One reason for the prominence of LTL as a specification language is the simple way in which it relates to natural language. For example, consider the natural language requirement for a traffic light controller (TLC): *whenever light is red, light should turn green within two minutes*. The following conversion steps convert the requirement into an MTL requirement.

1. Always when light is red then light should turn green within two minutes.
2. Always (if light is red, then light should turn green within two minutes)
3. Always (light is red implies that eventually light should turn green within two minutes)
4. Always (LightColor==RED Implies Eventually $_{\text{seconds}<120}$ LightColor == GREEN)

REM Tools: Code Generators and Monitors

The two primary categories of executable specification tools are code generators and REM tools (monitors). Code generators generate source code in a programming language such as a Java, C or C++, from formal, LTL specifications. While a conventional program can handle propositional logic, it cannot deal with higher forms of logic such as FOL, LTL, or MTL. For example, writing LTL inside a C program will result in compilation errors. Therefore, an often-used solution embeds high-level specification requirements inside program comments.

For example, the following C program contains an embedded MTL assertion for a TLC (written with syntax from [4]) asserting that *for 100 milliseconds, whenever light is red, camera should be on*:

```
void tlc(int Color_Main, boolean
      CameraOn) {
… /* Traffic Light Controller
      functionality */
/* TRBegin
TRClock{C1=getTimeInMillis()} //
      get time from the OS
TRAssert{ Always({Color_Main ==
      RED} Implies Eventually_ C1
      <1000_{CameraOn == 1})
      } =>
// Customizable user actions
{printf("SUCCESS\n");printf("FAIL\n");
      printf("DONE!\n");}
TREnd */
} /* end of tlc */
```

An executable specifications code generator generates code that replaces the embedded LTL/MTL assertion with real C code, which executes in process with the rest of the TLC, i.e., as part of the underlying TLC application. The generated code can also be used for formal specification-based exception handling [5].

In contrast to code generators, REM monitors (e.g., [6, 7]) monitor assertions in a stand-alone process often on a remote machine. It uses Hyper Text Transfer Protocol (HTTP), sockets, or serial communication to interface with the client application. To monitor, these tools either generate special, out-of-process source code using a code generator or use mathematical tools such as rewriting systems. The following list describes desired properties of remote monitors:

1. **Monitoring online, namely, no post-mortem processing is used.** A counter example would be to store all events in a database and use a structured query language (SQL)-based method to query those tables at a later time. The motivation for this requirement is that no expected termination time for the underlying application (e.g., security application) should be assumed. With no expected termination time, the size of the stored history trace information will be monotonically increasing and unbounded, which is unacceptable in most cases.
2. **Low impact**. It is desirable for a REM tool to not actively interrogate the client application (e.g., the banking system in the R2 example in the "Security Applications" section). Rather, it should passively listen to an event stream pertaining to basic propositions such as *deposit-occurred* or *balance<0*, which is sent to the REM tool from the client application. Having a low-impact REM tool increases the likelihood of acceptance by commercial and security-related organizations. For example, a bank is typically unwilling to be actively interrogated by a third-party tool but might agree to voluntarily, on its own terms and conditions, send out limited information to a third party such as a REM monitor.
3. **Powerful and flexible rules language.** Formal specifications need to capture real-life patterns and concerns such as real-time constraints while being syntactically close to natural language. The LTL satisfies these requirements; a large body of research points to its expressiveness and usefulness as a specification language. The MTL adds real-time constraints to LTL specifications. Time series constraints are also supported [8].

Run-Time Monitoring of Safety Critical Systems at NASA

NASA's Jet Propulsion Laboratory has used REM to verify the fault protection subsystem of the Deep-Impact spacecraft [9]. The level of fault protection provided by this system is single-fail-operational (the system has the capability to recover from a single fault and continue its mission). Multiple faults are handled sequentially where only one response is active at a time.

The fault protection provides monitoring and system-level responses to faults detected onboard the spacecraft. Other missions that have used this level of fault protection include Galileo, Mars Pathfinder, and DS1. Deep-Impact continues with this legacy but has incorporated a core reusable portion called the Fault Protection (FP) engine. The FP engine is responsible for accepting all input symptoms from various monitors and generating the appropriate system level response. These responses may vary from a single command (reset a hardware device) to a long running sequence (orient the spacecraft to a safe altitude).

The FP engine handles faults by priority-based input queues. The FP engine ensures that responses are handled in an orderly fashion and are not run unnecessarily (triggered by an overly sensitive monitor). After each response is run through to completion, the FP engine clears the fault and sends a cleanup signal back to the offending monitor.

"Formal specifications need to capture real-life patterns and concerns such as real-time constraints while being syntactically close to natural language."

REM was used to validate the FP engine software while in the development phase in executable form although not mature and robust; this led to the possibility of uncovering latent bugs in the software. Many aspects of the FP application exist that lend themselves well to a runtime verification approach. For example, during runtime the developer/tester may be unaware of inconsistencies within the internal state of the FP engine. Faults may be locked into a state where they are unable to trigger a response. Due to the nature of the application, there are hundreds of possible symptoms that may be reported to the FP engine, and dozens of possible faults that can trigger responses. It would be difficult for a test engineer to be constantly checking internal state consistency. Using the runtime verification approach, the REM monitor executed in parallel with the application, alerting the user to any violation of pre-defined correctness properties.

Security Applications[3]

Consider the following two airline security-related temporal pattern rules, both concerned with detecting a foreign national male passenger with a student visa flying to the Harrisburg International Airport near the Three Mile Island nuclear power plant:

- R1. Detect such a passenger if he has traveled to the Middle East at least once within a year of obtaining his student visa.
- R2. Detect such a passenger if he has traveled to the Middle East at least once within a year of obtaining his student visa and he received two or more direct deposits from non-US banks within the last year.

Both rules describe temporal patterns that contain potentially discernable elements from an airline security system operating automatically and in real-time. The two primary methods for performing such temporal pattern detection are *offline* and *online*, as described in the sequel.

The Federal Aviation Administration has an automated profiling system originally termed Computer Assisted Passenger Screening (CAPS) [10] that relies upon the data in each Passenger Name Record. This profiling system is being upgraded to access a more extensive range of data. The upgrade, CAPPSII, will profile airline passengers based on secret criteria to identify potential terrorists. Personal information about passengers may additionally include that from the Immigration and Naturalization Service (INS, now U.S. Immigration and Customs Enforcement), law enforcement, and customs. Having such history information stored in the system, or in constituent subsystems, enables SQL-based implementation of a temporal pattern rule such as R1. We call such an implementation *offline* because it relies on storing and querying historical information. An offline solution induces the following three impact consequences:

1. Temporal historical information is stored within the system (e.g., within CAPPSII and/or its constituent subsystems).
2. Temporal and non-temporal pattern detection is initiated by the security-related query, querying the constituent resources at will. We regard this as a high-impact solution because a temporal query is initiated from outside the original scope of the queried system, thereby impacting the performance of the queried system. For example, a potential INS subsystem of CAPPSII is impacted by repeated external queries from CAPPSII proper which, sooner or later, will degrade the INS system performance. Performance

degradation will occur because of the actual query processing forced on the INS subsystem and because of the fact that as time progresses, temporal queries might query monotonically increasing data-sets of historical data.
3. In addition to performance issues, CAPPSII and its constituent subsystems need to agree on a shared data representation for merging query results from multiple subsystems (e.g., merging INS and law enforcement query results).

This article is concerned with *low-impact*, *online* temporal pattern detection. It uses REM to detect temporal patterns without using historical data (i.e., it is online), and without querying the underlying application (i.e., it is low-impact). The only communicated information it requires from the underlying application (e.g., CAPPSII constituent subsystems) are Boolean messages for basic propositions such as *deposit of more than $1,000 was made to account of SSN=222 11 2222*.

While pattern rule R1 described earlier is programmable within the suggested CAPPSII framework, R2 requires extension, which includes banking information. Such an extension however will not lend itself to offline temporal pattern detection methods for the following reasons:

1. The banking data systems only store historical/temporal information for a limited duration (e.g., three months). The industry is unlikely to make any significant change to this policy.
2. Banking data systems are not likely to permit high-impact, CAPPSII-initiated queries because of the performance consequences discussed earlier as well as their own security need to be in full control over any content query.

In contrast, a REM temporal pattern detection method, being online and low-impact, can be used in tandem with CAPPSII while supporting extensions that support rules such as R2.

Conclusion

Executable specification methods have been effectively used to verify safety-critical systems at NASA. They enjoy the power and accuracy of formal specifications, yet are easy to use. They also enable requirement simulation prior to implementation. In addition, these appealing properties of executable specification methods lend themselves to non-verification applications such as monitoring temporal rules within security applications.◆

References

1. Manna, Z., and A. Pnueli. "Verification of Concurrent Programs: Temporal Proof Principles." Proc. of the Workshop on Logics of Programs. Springer Lecture Notes in Computer Science Vol. 1981: 200-252.
2. Pnueli, A. The Temporal Logic of Programs. Proc. of 18th IEEE Symposium on Foundations of Computer Science, 1977: 46-57.
3. Chang, E., A. Pnueli, and Z. Manna. Compositional Verification of Real-Time Systems. Proc. of 9th IEEE Symposium on Logic in Computer Science, 1994: 458-465.
4. Drusinsky, D. "The Temporal Rover and ATG Rover." Proc. of Spin2000 Workshop. Springer Lecture Notes in Computer Science Vol. 1885: 323-329.
5. Drusinsky, D. "Formal Specs Can Handle Exceptions." CMP Embedded Developers Journal Nov. 2001: 10-14.
6. Drusinsky, D., and M. Shing. Verification of Timing Properties in Rapid System Prototyping. Proc. of Rapid System Prototyping Conference, 2003.
7. Havelund, K., and G. Rosu. Monitoring Programs Using Rewriting. Proc. of IEEE Conference on Automated Software Engineering, 2001.
8. Drusinsky, D. "Monitoring Temporal Logic Specifications Combined With Time Series Constraints." The 4th Joint Workshop on Formal Specifications of Computer-Based Systems, 2003, extended abstract.
9. Drusinsky, D., and G. Watney. "Applying Run-Time Monitoring to the Deep-Impact Fault Protection Engine." The 27th IEEE/NASA ICECCS Workshop, 2002.
10. Fobes, J.L. Computer Assisted Passenger Screening (CAPS), DOT/FAA/AR-96/38, 1996.
11. Drusinsky, D., and J.L. Fobes. "Real-Time, Online, Low-Impact, Temporal Pattern Matching." The 7th World Multiconference on Systemics, Cybernetics, and Informatics, 2003.

Note

1. Propositional logic is a mathematical model that allows us to reason about the truth or falsehood of logical expressions.
2. Professor Amir Pnueli, a longtime advocate of temporal logic, is the 1996 Turing award winner for his "seminal work introducing temporal logic into computing science and for outstanding contributions to program and system verification."
3. Published in [11].

About the Author

Doron Drusinsky, Ph.D., is an associate professor of computer science at the Naval Postgraduate School, Monterey, Calif. His primary research interests are run-time monitoring and verification of safety-critical software systems. Drusinsky worked for Sony from 1988 until 1993 when he founded R-Active Concepts and authored BetterState, a UML statecharts design tool. BetterState was acquired by ISI-WindRiver Systems in 1997. Doron established Time Rover, Inc. and authored the Temporal Rover and DBRover formal verification tools. He has a Bachelor of Science from the Technion, Israel, and a doctorate from the Weizmann Institute.

Code CS/Dd
Naval Postgraduate School
Monterey CA, 93943
Phone: (831) 656-2168
Fax: (831) 656-3225
E-mail: ddrusins@nps.navy.mil

J.L. Fobes, Ph.D., conducts research and development for the U.S. Department of Homeland Security's Transportation Security Administration while serving at the Naval Postgraduate School, Monterey, Calif., as the Transportation Security chair. Formerly he was an associate professor at California State Los Angeles, conducted research for the U.S. Army Research Institute, was the chief of Human Factors for the U.S. Army Operational Test and Evaluation Agency, and the Federal Aviation Administration's program director for Aviation Security Human Factors. Fobes has a doctorate from the University of Arizona and a post doctorate from the California Institute of Technology, Pasadena.

Naval Postgraduate School
Monterey CA, 93943
E-mail: jlfobes@nps.navy.mil

Executable and Translatable UML[1]

Stephen J. Mellor
Mentor Graphics Corp.

Executable and Translatable Unified Modeling Language (xtUML), which is a profile of UML, adds a standard execution model for a tractable subset of the language so developers can formally test models to reduce defect rates from early execution of a target-independent application model before decisions have been made about implementation technologies. xtUML separates application and software architecture design so that each can evolve, be maintained, modified, and reused separately and concurrently. This yields significant reductions in development cost and time to market. In xtUML, design is expressed as a set of transformation rules, providing automatic model translation. The transformation rules are applied either uniformly or to model elements marked to indicate which rule to apply. This allows optimized patterns to be propagated throughout the code, providing powerful performance tuning and resource optimization. It also allows for retargeting models to different implementation technologies as they change. This article describes these fundamental ideas behind xtUML, and how they work in practice.

As its key idea, Executable and Translatable Unified Modeling Language (xtUML) separates application and software architecture design and weaves them together only at deployment time via the following:

- Application models capture what the application does clearly and precisely. The models are executable, including details relevant to the application but independent of the software platform (i.e., design and implementation details).
- Software architecture designs – defined in terms of transformation rules and execution engine components – are incorporated by a generator that produces code for the target system. The software architecture designs are completely independent of the applications they support.
- A generator, which may be human, weaves the application models and the execution engine components resulting in 100 percent complete code for modeled components.

The complete separation of the software architecture design from the application models supports concurrent design of the application and the software architecture, compressing the development schedule and time to market.

These benefits accrue partly as a result of simplifying the tasks of *analysis* and *design* because each can be carried out separately. In particular, design is the definition of a set of transformations that can be applied to the various analysis elements. Each design transformation rule thus applies to all matching patterns in the application, significantly simplifying the design task. This also enables automation, which yields greater benefits. For that reason, the article assumes automation here.

xtUML Notation

xtUML defines a carefully selected, streamlined subset of UML to support the needs of execution- and translation-based development, which is enforced not by convention but by execution: Either a model compiles, or it does not.

The notational subset has an underlying execution model. All diagrams (e.g., class diagrams, state diagrams, activity specifications) are *projections* or *views* of this underlying model. Other UML models that do not support execution such as use-case diagrams may be used freely to help build the xtUML models.

> *"... just as programming languages conferred independence from the hardware platform, xtUML confers independence from the software platform, which makes xtUML models portable across multiple development environments."*

The essential components of xtUML are illustrated in Figure 1 (see page 20), which shows a set of classes and objects that use state machines to communicate. Each state machine has a set of actions triggered by the state changes in the state machines that cause synchronization, data access, and functional computations to be executed.

A complete set of actions makes UML a computationally complete specification language with a defined *abstract syntax* for creating objects, sending signals to them, accessing data about instances, and executing general computations. An action language *concrete syntax*[2] provides a notation for expressing these computations.

An xtUML model with actions is not a blueprint to be rewritten or filled out by programmers, but an executable specification. The difference between an ordinary programming language and a UML action language is analogous to the difference between assembly code and a programming language. They both completely specify the work to be done, but they do so at different levels of language abstraction. Programming languages abstract away details of the hardware platform so you can write what needs to be done without having to worry about things such as the number of registers on the target machine, the structure of the stack, or how parameters are passed to functions. The existence of standards also makes programs portable across multiple machines.

xtUML allows developers to model the underlying semantics of a subject matter without having to worry about such things as the number of processors, the data-structure organization, or the number of threads. In other words, just as programming languages conferred independence from the *hardware* platform, xtUML confers independence from the *software* platform, which makes xtUML models portable across multiple development environments.

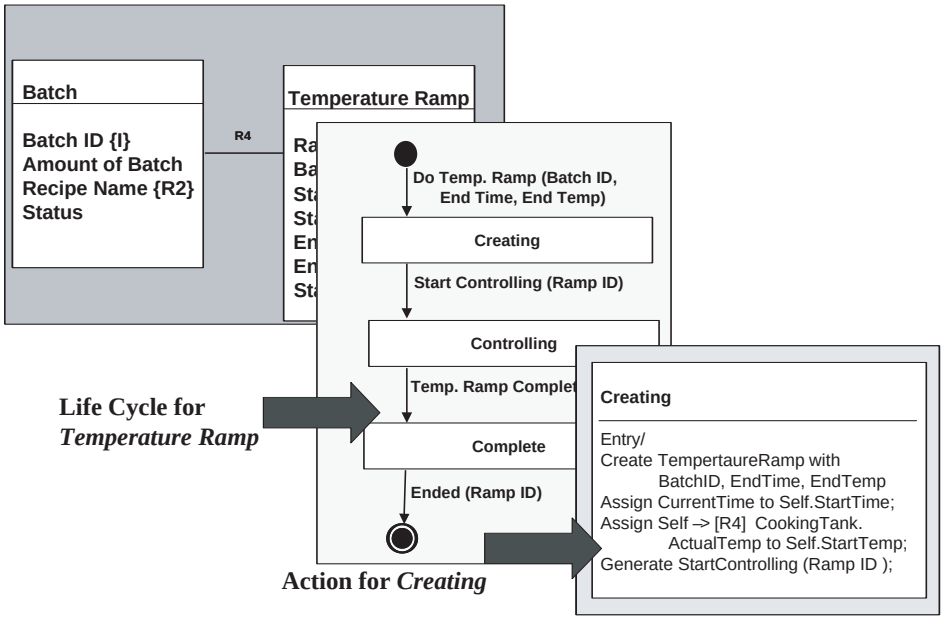


Figure 1: *The Structure of an xtUML Model*

xtUML Dynamics

Figure 1 shows the static structure of xtUML, but a language is not meaningful unless there is a definition of the dynamics, or how the language executes at run time. To execute and translate, xtUML must have well-defined execution semantics – and it does. xtUML has specific, unambiguous rules regarding dynamic behavior, stated in terms of a set of communicating state machines, the only active elements in an xtUML program.

Each object and class (potentially) has a state machine that captures the behavior over time of each object and class. Every state machine is in exactly one state at a time, and all state machines execute concurrently with respect to one another. A state machine synchronizes its behavior with another by sending a signal that is interpreted by the receiver's state machine as an event. On receipt of a signal, a state machine fires a transition and executes an activity, a set of actions that must run to completion before the next event is processed.

State machines communicate only by signals, and signal order is preserved between sender and receiver instance pairs. The rule simply enforces the desired sequence of activities. When the event causes a transition in the receiver, the activity in the destination state of the receiver executes *after the action that sent the signal*. This captures desired cause and effect in the system's behavior. It is a wholly separate problem to guarantee that signals do not get out of order, links fail, etc., just as it is a separate problem to ensure sequential execution of instructions in a parallel machine.

Each activity comprises a set of actions such as a computation, a signal send, or a data access. The semantics of these actions are defined so that data structures can be changed at translation time without affecting the definition of computation – a critical requirement for translatability. The actions in each activity execute concurrently unless otherwise constrained by data or control flow, and these actions may access data of other objects. It is the proper task of the modeler to specify the correct sequencing and to ensure object data consistency.

The application model, therefore, contains the details necessary to support application model execution verification and validation, independent of design and implementation. No design details or code needs to be developed or added for model execution: Formal test cases can be executed against the model to verify that application requirements have been properly addressed.

Those are the rules, but what is really going on is that xtUML is a concurrent specification language. Rules about synchronization and object data consistency are simply rules for that language, just as in C++ we execute one statement after another and data is accessed one statement at a time. We specify in such a concurrent language so that we may *translate it* onto concurrent, distributed platforms, as well as fully synchronous, single-tasking environments.

At system construction time, the conceptual objects are mapped to threads and processors. The generator's job is to maintain the desired sequencing specified in the application models, but it also may choose to distribute objects, sequentialize them, even duplicate them redundantly, or split them apart *so long as the defined behavior is preserved.*

Model Compilers

A model compiler automatically generates target-optimized, 100 percent complete code from models. A model compiler comprises two main components. First, there is an execution engine that supplies the execution infrastructure such as storage schemes, action invocation, and signal sending. An execution engine is a specific set of reusable components that, when taken together, are capable of executing an arbitrary executable UML model. The execution engine will therefore contain ways of storing instances in some form, possibly as objects, but not necessarily; some way of invoking an action; some way of sending signals; some way of reading an attribute; and so forth. The selection of the elements in the execution engine determines the system properties such as concurrency and sequentialization, multiprocessing and multitasking, persistence, data organization, and data structure choices. These choices, together with the pattern of usages in the application, determine the performance of the system.

Second, a set of *archetypes* specifies how to translate an application model into code. Archetypes are a formalization of the design patterns and translation rules. The archetype describes when it should be used, the set of patterns to be applied in code generation, and how model components will be populated or utilized to build code. (Archetype examples are provided in the next section.) The combination of translated application code, legacy code that is linked, and the execution engine constitutes the runtime system.

The archetypes use a generator to traverse an arbitrary repository and produce text. The repository contains the meaning of application model, distinct from the diagrams. (The repository will maintain graphical information too, but that is not of concern for generation.) The logical structure of the repository (the *metamodel*) mirrors the semantic rules described in the previous sections, including the semantics of actions – which means that the repository contains the entire, detailed application model.

The metamodel is a model of xtUML using UML. It has classes such as Class,

Attribute, Event, State, Action, CreateAction, and ReadAction – all the concepts we have discussed. When we draw a class such as Batch in a developer model using an xtUML-aware model building tool, it creates an instance of Class, with data describing the class so created such as a Name (Batch), a description, and the like. Similarly, when we create an attribute amount of Batch, this creates an instance of Attribute with name (amount), the class it describes (a reference to Batch), and a type (quantity).

The generator is a translation engine that extracts application model information, interprets the archetypes, and performs the mapping of model components to generate complete code. (Recall that the repository contains the actions too.) The partitioning of model compilers into these pieces streamlines their customization, construction, and maintenance. Changes and additions can be made to the archetypes or run-time library without having to contend with the details of generator or repository management.

Generator Operation

The generator and the archetypes constitute a compilation environment. When generating code, the generator extracts information from the application model. The generator then selects the appropriate archetype for the *to-be-translated* model element. The information extracted from this model is then used to *fill in the blanks* of the selected archetype. The result is a fully coded model component.

The archetypes are applied either uniformly to certain kinds of model elements (all classes, say), or to model elements that have been marked to indicate which rule to apply. For example, a class could be marked to indicate the processor in which it resides, or a state chart could be marked to show which storage scheme (a list or a table) to use, and so on. Using marks provides complete control over the output and enables performance optimization at any level.

Population of an archetype commonly requires invocation of other archetypes. These newly invoked archetypes, in turn, often invoke other archetypes. The creation of code, for what appears to be one model element, can ultimately involve several nested layers of archetypes for multiple model elements. This is fully automated by the generator. This simple approach is incredibly powerful for real-life applications.

The simple archetype in Table 1 generates code for private data members of a class by selecting all related attributes and iterating over them. All lines beginning with a period (.) are commands to the generator, which traverses a repository containing the executable model and performs text substitutions.

```
.Function PrivateDataMember( class class )
.select many pdmS from instances of Attribute related to class;
.for each pdm in pdmS
${pdm.Type} ${ pdm.Name};
.endfor
```

Table 1: *Simple Archetype*

Archetype	Generated Code
.select many *stateS* related to instances of *class*->[R13]StateChart ->[R14]State where (isFinal==False); public: enum states_e {NO_STATE=0, .for each *state* in *stateS* .if (not last *stateS*) ${*state*.Name}, .else NUM_STATES = ${*state*.Name} .endif; .endfor; };	public: enum states_e {NO_STATE = 0 , Filling, Cooking, NUM_STATES = Emptying };

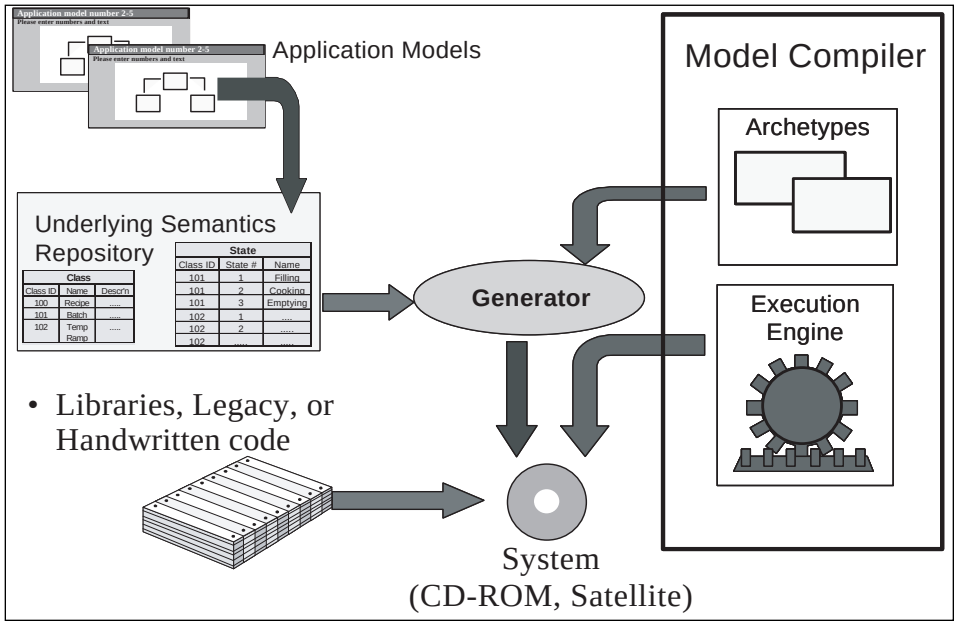
Table 2: *Example Archetype*

${pdm.Type}, as shown in Table 1, recovers the type of the attribute and substitutes it on the output stream. Similarly, the fragment ${pdm.Name} substitutes the name of the attribute. The space that separates them and the lone semicolon (;) is just text, copied without change, onto the output stream.

In the more complete example in Table 2, the archetype uses italics for references to instances in the repository, underlining to refer to names of classes and attributes in the repository, and noticeably different capitalization to distinguish between collections of instances versus individual ones.

You may wonder what the produced code is for. It is an enumeration of states with a variable num_states automatically set to be the count for the number of elements in the enumeration. (There is a similar archetype that produces an enumeration of signals.) The enumerations are used to declare a two-dimensional array containing the pointers to the activity to be executed. You may not like this code, or you may have a better way to do it. Cool. All you have to do is modify the archetype and regenerate. Every single place where this code appears will then be

Figure 2: *Components of an Automated Solution*



changed. Propagating changes this way enables rapid performance optimization.

While the generated code is less than half the size of the archetype, the archetype can generate any number of these enumerations, all in the same way, all equally right or wrong.

Because the archetype is a data access and text manipulation language, it can be used in conjunction with the generator to generate code in any language: C, C++, Java, or Ada, and, if you know the syntax, Klingon. We have used xtUML to generate Very High Speed Integrated Circuit Hardware Description Language.

System Construction

Figure 2 shows how the pieces fit together. To build a system, developers build xtUML models that specify the desired behavior of the application, and buy a model compiler comprising a set of archetypes and an execution engine. The compilation process proceeds in two phases. First, the archetypes traverse the model as stored in the repository to produce source code. Second, the generated code is compiled with the execution engine library and any handwritten, legacy, or library code. The result is the system.

Examples

xtUML has been used on over 1,400 real-time and technical projects, including life-critical implanted medical devices, Department of Defense flight-critical systems, 24x7 performance-critical fault-tolerant telecom systems, highly resource-constrained consumer electronics, and large-scale discrete-event simulation systems.

One telecommunications switch project with an in-house model compiler generated in excess of 4 million lines of C++. One hundred percent of the modeled code was generated, comprising over 80 percent of the total system code. The system was extremely real-time, fault-tolerant, and used over 1,000 distributed processors [1].

A consumer electronic project compared handwritten code with a model compiler. The model compiler generated faster code than the handwritten version, though it was slightly bigger. This difference was traced to different choices made for caching variables. Both the handwritten code and the generated code met performance and size constraints [2].

A joint forces wargaming system built xtUML models of the maritime portion of the battlespace and translated them into C++ that runs on an high level architecture-based distributed discrete event-simulation engine. The target platforms were UNIX workstations and Windows boxes. They used a customized version of MC-2020 as the base for the model compiler. One portion of the simulation uses a special purpose simulation language generated by archetypes. The model compiler was derived from a C++ model compiler with similar system characteristics.

Another organization that has built its own model compiler, for sale, with sophisticated transaction safety and rollback features reports between seven and 10 lines of generated C++ for each line of action language. More importantly, all that delicate code is known to be correct; it is not handcrafted by fallible, or worse, *creative* coders.

For a completely worked out, publicly available example model, see "Executable UML: The Case Study" [3].

xtUML Capabilities

xtUML provides a unique opportunity to accelerate development and improve the quality, performance, and resource utilization of real-time, embedded, simulation, and technical systems. The approach provides for the following:

- Fully customizable translation generating 100 percent complete, target-optimized code.
- Reduced defect rates from early execution of target-independent application models by an average of 10 times (*not* 10 percent).
- Accelerated development of products with multiple releases, growing or changing requirements, and families of products.
- Concurrent design and application analysis modeling to compress project schedules.
- Powerful performance tuning and resource optimization.
- Effective, practical reuse of target-independent application models.
- Effective, practical reuse of application-independent designs.
- Reduced maintenance costs and extended product lifetimes.

This article described the fundamental ideas behind xtUML and how it works in practice. These ideas are more fully described in [4].◆

References

1. Case, J., and John R. Wolfe. "Modeling Accelerates Optical Networking System Development." Project Technology Inc. <www.projtech.com/pdfs/success/tellabs.pdf>.
2. Yamaguchi, Minoru. "Recursive Design for Real-Time Embedded Systems." Sony Corp. 2001 <www.projtech.com/pubs/confs/2001.html>, then access yamaguchi.zip.
3. Starr, Leon. Executable UML: The Case Study. 2nd ed. Model Integration LLC, 21 Feb. 2001.
4. Mellor, Stephen J., and Marc J. Balcer. Executable UML: Foundation for Model-Driven Architecture. 1st ed. Addison-Wesley Professional, 15 May 2002.

Note

1. Parts of this article were derived from the draft for "MDA Distilled: Principles of Model-Driven Architecture" by S.J. Mellor, K. Scott, A. Uhl, and D. Weise, Addison-Wesley, 2004.
2. BridgePoint provides an Object Action Language that is compliant with the abstract syntax standard, but there is presently no action language (notation) standard.

About the Author

Stephen J. Mellor is chief scientist of the Embedded Systems Division at Mentor Graphics. He is an internationally recognized pioneer in creating effective engineering approaches to software development. Mellor is active in the Object Management Group, chairing the consortium that added executable actions to the Unified Modeling Language (UML), and is now active in specifying model-driven architecture (MDA). He is chairman of the "IEEE Software" industrial advisory board and is a signatory to the "Agile Manifesto." He is author of "Executable UML: A Foundation for Model-Driven Architecture," "MDA Distilled," and publisher of the 1985 Ward-Mellor trilogy "Structured Development for Real-Time Systems," and the first books defining object-oriented analysis in 1988. Mellor co-founded a company focused on tools to execute and translate xtUML models that is now a part of the Embedded Systems Division of Mentor Graphics where he is chief scientist.

E-mail: stephen_mellor @mentorg.com

What You Don't Know Can Hurt You

Douglas A. Ebert
McKesson Corporation

This article provides senior managers with a methodology to develop a metrics program that will form a basis for management decisions. It presents a series of questions a senior manager should ask to address business needs, rather than just getting informational briefings.

There is an old accounting adage that states, "You can't manage what you can't measure." In the development world we certainly take this to heart. Walk into any boardroom of any information systems development group and you will find remnants of charts, briefings, status reports, and analyses of all types – and yet, in the 2003 version of the Standish Group's "Chaos Chronicles" [1], only 34 percent of all information technology projects are labeled successful!

Maybe we just do not have enough metrics. Some time ago I had a discussion with a project manager working on a large development project, the manager estimated that he and his subordinate team leaders spend around 30 percent of their time preparing reports aimed at higher levels within the company. These reports culminate in monthly senior stakeholder meetings attended by the highest-ranking members of the business, yet 15 months into the project, the manager cannot recall any directions or decisions coming from these meetings. This void is in spite of the fact that the project will come in over a year late and that early deliverables have received unfavorable comments about their quality!

As senior managers, we have gone astray somewhere in our metrics programs. It is not enough just to be fed with data if that data is not able to drive decisions. We could put the old accounting adage another way: "You don't measure because you want to know, you measure because you want to be in control!"

Standard Metrics Programs Quickly Become Stale

Most of us have a standard set of project reporting templates that we expect our development project managers to fill out periodically. These reporting systems evolve over time because senior leaders really do have a need to know what is going on. The problem is that without any more thought than this, managers default to asking for metrics that are readily available. This includes things like work hours or resources spent to date, defects uncovered so far, and the proximity to being on budget. Worse, without any insight into what we think this data shows, our project managers obediently and simply provide that data.

While perhaps interesting, this kind of information does not provide the senior manager with any basis of understanding about whether the project really is going well or poorly. Nor does it indicate if there is anything he or she can (or even needs to be able to) do about it. Thus, project reviews become informational briefings, not periodic checkups. Watts S. Humphrey says it best:

> By concentrating on the schedule, managers overlook the things they can influence. These are to start jobs promptly, provide adequate staffing, and ensure that the work is done in a disciplined and professional way. [2]

"As senior managers, we have gone astray somewhere in our metrics programs. It is not enough just to be fed with data if that data is not able to drive decisions."

What we need is a program that will link the requested metrics to business objectives and provide "objective results that can be used in making informed decisions, and taking appropriate corrective actions" [3]. As an important part of this program, our project managers need to know why we think we need this data. Knowing what we think we are getting, product managers will be able to suggest their own methods and reporting tools. This helps build a consistent approach and provides a basis for mentoring as well. Building a set of metrics consists of asking the following five basic questions.

1. What Do You Need To Know?

This is not a trivial question and is an essential step since it forms the basis for the entire metrics program. Rather than treating every project alike, take the time to sit down with your project manager to define information needs based not only on the specific nuances of the project, but also on the need to be alerted if decisions are required.

For example, it is unwise for a senior leader to fall into the trap of making a simple needs statement like, "I need to know if we are on schedule." A better needs statement would be something like, "I need to know that I'll stay on schedule." The difference in these two statements is not that subtle. The first statement is a simple snapshot that may let the senior manager sleep that night, but does not tell him whether he should cancel his upcoming vacation time. More to the point, if the schedule is slipping, the senior leader has no basis to understand why it is slipping and what can be done about it.

Then again, each project is a proving ground to see if you learned any lessons from the last project. As a senior manager, you always have a need to know that with each project you are getting better at something – whether that something is better efficiency, productivity, or just plain quality. With each project, target metrics that will help evaluate improvement measures, either to establish a baseline or assess improvements. Need statements for this category could look like this: "I need to know that my quality improvements are meeting the objective," or, "I need to know the volatility of requirements during each stage of development."

It is interesting that when I was a project manager, my senior managers would regularly ask me, "What are you doing differently?" But rarely would they demand that I demonstrate my changes had any effect! It seemed just causing the disruption associated by introducing new processes met their expectations.

2. What Questions Do You Need to Ask?

For each of your need-to-know items, you have to design the questions correctly to get useful information – this is not as simple as it might seem. Again, many senior managers ask for simple data points such as the current project master timeline. The project master timeline is data; being able to demonstrate that the project kept to that timeline on purpose rather than by accident requires analysis.

For example, let us take the following case: "I need to know that I'll stay on schedule." One question you might want to ask would be, "What is my burn rate of resources for each phase/task of a project compared to my estimates?" It should be noted that to answer this question, a project manager will still need to present the traditional data points of milestones and resources consumed in a period of time.

Be watchful for events inside the project that might create problems later. This not only includes scope creep, where new requirements are being added, but also scope volatility. A programming effort in which I was involved was right on schedule for the first eight months, although during the last month a change in direction caused changes in almost 15 percent of the requirements. The needed rework did not really show up as delays for another two months and came as a complete surprise to senior managers.

Here is a hint: One good place to look for questions is to challenge planning assumptions. Assumptions made during program planning are excellent informational needs for the measurement process. If the assumption is not realized, then many of the resulting schedule and resource plans may need to be re-examined, or replanned. [4]

Simplistically, if the project plan assumes 70 percent availability of staff, that becomes a key metric to predict the final project outcome.

3. Why Do You Think This Is the Right Question?

This is the introspective part: Challenge yourself about each question you believe you would want to ask. This requires you to think through to the *so what* of whatever metrics you may receive – to think through what you might do with the data and what management actions you might take. This step also prepares you to defend your demand for these metrics.

For example, you may explain your rationale as this: "Knowing the resource burn rate compared to estimates will give me a glimpse into the solidness of the project plan. If our estimates are consistently low, I may have to reset project expectations or assign additional resources to your team."

"One good place to look for questions is to challenge planning assumptions. Assumptions made during program planning are excellent informational needs for the measurement process."

Another example, if your need-to-know is that we have been meeting requirements of our quality program. A need-to-ask question could be: "What is the trend of SQA [software quality assurance] violations, by type?" Why would you want to know this? Humphrey says it best: "When software projects fail, it is usually because a manager did not insist that the work be done in the right way" [2].

Here is another example: "How do the hours expended to address customer-related problems compare to my budget?" Why would you want to know this? Well, unless you have the luxury of a separate, dedicated support group, surges in customer-related efforts often rob your project of essential resources. Remember, if you wanted to have a good feeling that you would stay on target, this is a question you may need to ask.

You may wish to discuss your rationale with your managers. This will give them insight to your needs and concerns and allow them to provide their own input. Do not misunderstand, though; this is not a voting situation. Just because there is no current process to answer your questions does not mean you should not get the answers.

4. What Measures Would Provide Input to Management Actions?

In most cases, you may already have some idea of the questions you need to ask. Stop and refine those questions to ensure you have enough detail to take action. For instance, knowing about SQA violations will give you an overall feeling about how well your process programs are being followed. However, having these violations reported per project step or phase may highlight where your retraining efforts may provide the biggest impact.

This polishing question could also be asked like this: "Before I take action to correct a problem, what other information would I need?" If you knew the distribution of defects per module, reported by severity, that would be more useful in a project review than just hearing something like, "We're sure seeing a lot of problems!"

This is the time to look for collaborative or explanatory evidence. In this last example, it would be interesting to compare SQA violations to defects per module as you would expect to see a correlation between the two. However, if SQA violations are not accompanied by defects, it could mean your SQA processes are a little overzealous! On one project, a rash of SQA violations was reported during inspections of requirements specifications. Upon further investigation, we discovered that the documents associated with the requirements program had recently been automated, yet the SQA checklist still required physical signatures by both author and checker.

5. What Is My Objective for Each Metric?

Put another way, this last question sets an acceptable range of values for each metric. It is important to set up each of these trip points ahead of time. Do not wait to get a report before you have to ask the question, "Should I be worried about this?"

For example, you may wish to call a more in-depth schedule review if the resources allocated are greater than 10 percent of the estimated burn rate. Or, perhaps you have set a goal of decreasing SQA violations in the planning phase by 20 percent. You must choose the acceptable range of values based on specific circumstances of the project, including the broader standards that may be established by your company and the maturity of your project members.

This last question also provides a kind of acid test. If there is no unacceptable

value – if no actions are forthcoming regardless of the report – then this is data, and not a metric.

Final Words of Advice

Here are some final suggestions in building a set of metrics:

- Avoid the simple answers – they may be good data points but they are generally not helpful.
- Metrics can have a life of their own so avoid creating a monster! Start small and grow with time. Remember that collecting metrics consumes project resources.
- Make establishing a baseline a priority if you do not have one already.
- Listen after you ask questions. This encourages participation and mutual respect.
- Remember, the primary purpose of a metrics program is to support change! (As a corollary; metrics programs make very poor clubs.)
- Push the comfort zone. You need to know what you need to manage, not what information somebody is comfortable telling you.◆

References

1. The Standish Group International, Inc. CHAOS Chronicles. Vers. 3.0. West Yarmouth, MA.: The Standish Group, 2003 <www.standishgroup.com/chaos/toc/php>.
2. Humphrey, Watts S. Winning With Software. Boston, MA: Pearson Education, 2002.
3. Software Engineering Institute. Capability Maturity Model® Integration (CMMI®), Vers. 1.1. Pittsburgh, PA: Carnegie Mellon University, 2001.
4. Baxter, Peter. "Focusing Measurement on Managers' Informational Needs." CROSSTALK July 2002: 22-25.

About the Author

Douglas A. Ebert completed a U.S. Air Force career as a lieutenant colonel and is currently a vice president for strategic planning and partnerships with McKesson Corporation. In this capacity, he leads the adoption of emerging industry technologies and standardization efforts in developing information solutions for the healthcare industry. A speaker at technology conferences, he is also the chair of the Software Engineering Institute's Capability Maturity Model Integration® Interpretive Guidance Expert Group.

McKesson Corporation
1400 S. Wolf RD STE 200
Wheeling, IL 60090
Phone: (847) 495-1718
Fax: (847) 537-4866
E-mail: doug.ebert@mckesson.com

COMING EVENTS

October 2-9
Internet, Processing, Systems, and Interdisciplinary Research (IPSI) 2004
Sveti Stefan, Montenegro
http://montenegro.internetconferences.net

October 3-6
2004 IEEE Custom Integrated Circuits Conference CICC 2004
Orlando, FL
http://www.ieee-cicc.org

October 4-6
14th International Conference on Software Quality
Orlando, FL
http://software.asqquality.org/events.htm

October 6-7
IT Research Associates' 3rd Annual Six Sigma for Software Development
San Francisco, CA
http://www.frallc.com/infotech.asp

October 9-13
11th International Conference on Architectural Support for Programming Languages and Operating Systems
Boston, MA
http://www.eecg.toronto.edu/asplos2004

October 24-27
17th Annual ACM Symposium on User Interface Software and Technology
Santa Fe, NM
http://www.acm.org/uist

October 31-November 6
Association for Computing Machinery SIGSOFT 2004
Newport Beach, CA
http://www.isr.uci.edu/FSE-12/venue.html

April 18-21, 2005
2005 Systems and Software Technology Conference
Salt Lake City, UT
www.stc-online.org

Identifying Essential Technologies for Network-Centric Warfare

David Schaar
PRICE Systems, LLC

Network-centric warfare (NCW) is still a concept that is being defined; many find it too intangible for comfort. This is an attempt at materializing what, exactly, will be the important technologies for NCW.

Most people's first encounter with the term *network-centric warfare* (NCW) ought to set off their undefined-buzzword-that-sounds-fancy radar. It appears sufficiently generic an expression to encompass any computer-based warfighting system. It is true that there is no dictionary definition of the term. This calls for a clarification of the sense it will have in this article: *NCW is about leveraging existing information assets using an infostructure*.

Now, let us dissect this statement: The term infostructure is an amalgamation of the words *information* and *infrastructure* – it refers to the infrastructure used for information sharing. This could be anything from a long-wave military radio network to an office Local Area Network (LAN). The next keyphrase is *existing information assets*. This establishes that NCW is not about creating new information, but rather about using the information that is already in our possession. Finally, the word *leveraging* is of crucial importance: We are trying to make better use of what we already have. Based on these premises, NCW is about creating battlespace superiority through more efficient use of existing information.

The concept of NCW can be further illustrated by an example. Think of a situation where Army tanks, Navy ships carrying short-range missiles, and Air Force ground attack aircraft would be deployed to take out a mobile enemy command unit. Rather than each moving independently toward the target, they would use a common data network to coordinate their efforts. Each unit type has various sensors to track the target, and this data is fed into the data network. The data is processed into one single target reading that is returned to the units, rendering much more accurate positioning. As the units move in closer to the target, they all have the friendly tank positions plotted on their map displays to avoid friendly fire incidents. The Navy ships have real-time information on the location of the attack aircraft as the ships get ready to launch their missiles. Finally, when weapons are launched, all units receive continuous feeds on the status of the target to optimize impact.

Clearly this way of taking out the target is more likely to have favorable results at a lower cost compared to a situation where all units act independently. It is made possible through intelligent sharing of information.

> *"NCW [network-centric warfare] primarily focuses on ... enabling better awareness of the enemy and friendly forces – but also places emphasis on improving command and control and decision making as well as execution ... through improved communications systems."*

What will it take, technically, to achieve this battlespace superiority? This article attempts to materialize what, specifically, the crucial components are for making it possible to reap the benefits of NCW. The approach is general, not focusing specifically on the United States or any other military force.

The Functions in NCW

In their book "Network-Centric Warfare" [1], the authors identify three roles in the battlespace (*battlespace* as opposed to *battlefield* reflects the reality that today's battles are not necessarily fought in a single geographically delimited theater). These roles carry out the three main functions – or tasks – in the battlespace: 1) achieving battlespace awareness and knowledge, 2) command and control and decision making, and 3) execution.

These functions are carried out by the *roles* of sensors, decision-makers, and actors, respectively. The concept of NCW primarily focuses on the first of the functions – enabling better awareness of the enemy and friendly forces – but also places emphasis on improving command and control and decision making as well as execution, for instance, through improved communications systems.

An illustrative example of these functions is a forward-deployed reconnaissance squad determining the exact location of a target (*sensor*) and reporting this back to the command center. At the command center, the order is given (*decision-maker*) to an aircraft in the area to take out the target (*actor*).

At the other end of the spectrum, the three roles can be carried out by one and the same entity: An infantry soldier spots an enemy soldier (sensor), determines that he needs to attack the enemy soldier (decision-maker), and proceeds to fire his weapon against him (actor).

Analysis Based on the NCW Roles

Splitting the analysis along these functions is a good inroad to trying to determine what it will take for NCW to be a success. Of particular interest to this community is the first function: How to achieve battlespace awareness and knowledge. This function is not an effort to gather more information; remember, we are in the business of better using our *existing* information assets. Rather, it is an attempt to share and process information in the best way possible. The remainder of this article will consist of a closer look at the sensor function.

Battlespace Awareness and Knowledge Analysis Methodology

This research is an attempt at formalizing

the analysis of key technological areas for enabling NCW: How do we find the *hottest* information sharing nodes where exchange of sensor data is the most valuable? It presents a methodology for creating a relative ranking of the information sharing nodes. The following is a description of that methodology.

The key to the approach was to start with a complete model of the battlespace and from there, try to *zero in* on the most interesting areas from a NCW perspective. The chosen model would describe all the possible transactions between information nodes in the battlespace such as depicted in Figure 1. An information node was equated with any battlespace entity (aircraft carrier, fighter aircraft, armored personnel carrier). Using a ranking methodology, the most interesting of these nodes would be identified.

The basis for the graph in Figure 1 was a matrix that described all the possible information transactions. An information transaction involves an information supplier and an information recipient. The information supplier translates into a sensor, as described earlier, carried by some battlespace entity. The information recipient corresponds to any battlespace entity. For instance, there could be an aircraft carrying photoreconnaissance equipment (*information supplier*), transmitting image data to a ground unit charged with the task of taking out a target (*information recipient*).

By applying mathematical methods as described further on in this article, the value of each information transaction could be assigned a relative value, which in turn formed the basis for a graph with rankings.

Initially, a list of all main unit types in the battlespace from sources such as [2] was created. For each of these unit types, a list of all possible categories of sensors that could be carried was assembled. From this, a matrix (Figure 2) was compiled to describe the value of all possible information transactions between the entities in the list. This approach was inspired by John J. Garstka's method of representing information positions [3]. Along the x-axis of the matrix are listed all possible suppliers of information such as air radar carried by an attack aircraft or global positioning system (GPS) carried by an armored personnel carrier. Along the y-axis are all possible information recipients such as bomber aircraft or submarine.

Each element i_{jk} represents the value of information flow from element k to element j, for example, the value of information flow from attack aircraft air radar to a bomber aircraft. In the study, each element was assigned an integer value between 0 (low importance) and 2 (high importance) to determine the significance of the information transaction.

Additionally, each column was multiplied with a weighting factor to indicate the significance of the sensor being carried by a particular unit type – for instance to account for the fact that a surface warfare ship will not necessarily be carrying an air radar (low weighting), while an anti-submarine warfare aircraft is guaranteed to be carrying a sonar (high weighting).

This resulted in a 150 X 25 matrix. The values in this matrix were absolute; that is, they were all measured along the same scale but they had not been adjusted to reflect the relative importance between them. Having the values relative rather than absolute was key to being able to graph their significance. The absolute values were transformed into relative values through the following method:

- In the interest of keeping the results manageable, the matrix data was grouped into aggregate elements according to different rules for each analysis. For instance, one analysis was performed where aggregate groups of transactions based upon the sensor type (e.g., *air radar*) were created, and another analysis was done where groups based on the carrier unit type (e.g., *ground unit*) were compiled.

For illustrative purposes, we will look in more detail at the case where information suppliers were grouped according to the sensor type as well as carrier type.

Figure 1: *Information Transaction Graph*

One supplier group was *air radar* carried by air units. In this analysis, one information recipient group was *sea units*. The value to be calculated was thus the relative importance of information flow from air radar carried by an aircraft to a sea unit. The formula was the following:

$$\text{Relative Importance} = \frac{\sum \text{Matrix}_{jk}}{\text{Count}}$$

where,

***j* is an element of recipient group, *k* is an element of supplier group,and *count* is the total number of suppliers in the group.**

The outcome of this calculation is that the relative value was computed as the sum of all matrix elements for air radars carried by aircraft where the information recipient is a sea unit, divided by the number of air radars carried by aircraft.

A Java program was written for these calculations and for formatting the results into inputs for the Graphviz tool from AT&T [4]. Graphviz was then used to produce illustrative graphs such as the one in Figure 3 (see next page).

Figure 2: *Information Transaction Matrix*

$$\begin{array}{c} \\ \text{Information\_recipient} \end{array} \begin{array}{c} \text{Information\_supplier} \\ \begin{bmatrix} \text{Weight}_1 & \text{Weight}_2 & \text{Weight}_n \\ i_{1,1} & i_{1,2} & i_{1,n} \\ i_{2,1} & i_{2,2} & i_{2,n} \\ i_{m,1} & i_{m,2} & i_{m,n} \end{bmatrix} \end{array}$$

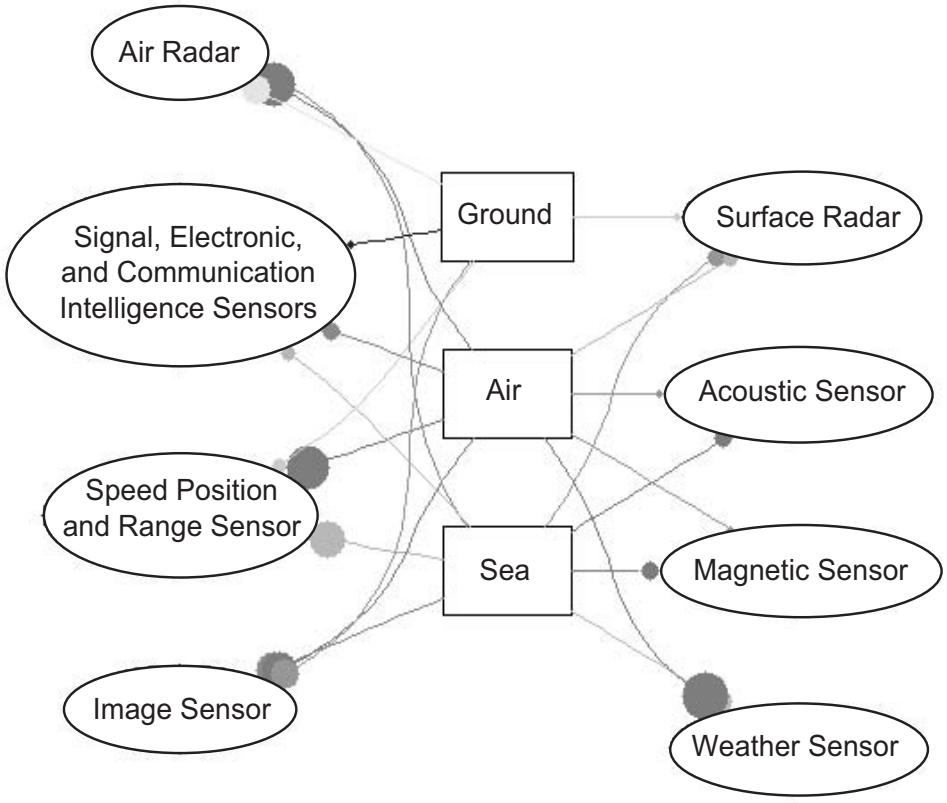


Figure 3: *Information Transaction Matrix*

Results

The graphing resulted in clearly distinguishable areas of interest. For information suppliers, four sensor types were distinctly at the top of the list:

1. Friendly unit positioning sensors.
2. Weather sensors.
3. Enemy positioning sensors.
4. Imaging sensors carried by aircraft.

An example of friendly unit positioning sensors is a GPS system. Friendly units receive a feed from the sensor on the unit's position. This creates a very high level of battlespace awareness of friendly units' whereabouts.

Weather sensors range from temperature sensors to weather radars – anything that may inform other units of the current weather conditions.

Enemy positioning sensors are typically radars. A fighter aircraft may receive a feed from other friendly aircraft tracking the same enemy unit. Creating a fused reading from the friendly aircraft's data indicating where the enemy unit is renders a more accurate positioning and, hence, better efficiency.

Imaging sensors carried by aircraft are sensors that capture either still or video imagery, either visible light or infrared.

On the information recipient side, air and sea units were deemed more interesting than ground units. This reflects the fact that these units typically carry more advanced processing systems and are able to take in information from more sources. It also illustrates the greater necessity for aircraft and ships to be oriented about the whereabouts of all friendly units as well as enemy units.

So what is the impact of these results? We can draw the conclusion that the four sensor types previously identified ought to be among the systems receiving particular emphasis in military acquisitions. Similarly, they are likely to be receiving particular attention by equipment manufacturers. Development of these systems is likely to be prioritized.

One area merits some extra thought from a software viewpoint: Looking particularly at sensor types one and three, data fusion – the art of taking readings on the same phenomenon from several sources and applying algorithms to generate a single, more accurate reading – will be of special interest. The type of data fusion in question here is referred to as *positional fusion* [5]. Three component tasks make up positional fusion:

1. **Data alignment:** Transforming sensor data into a common frame of reference.
2. **Parametric association:** Associating observations into groups that represent the same entity.
3. **State vector estimation:** Combining the observations that result from the same entity into a single estimation of the entity's position and velocity.

Clearly, this is a very processing-intensive area, with a focus on optimized software.

While it might be considered stating the obvious, it should also be pointed out that software areas of general relevance to NCW are, among others, network operating systems, network interface software, and communications applications.

The Next Step

In a look further down the line, one author [6] envisions a departure from direct connections between a sensor and the user. Rather than each unit carrying its own sensors, there would be so-called *data fusion nodes* to which both suppliers and consumers of information would connect. This would be a hub of sorts, with a task manager that, when a request for information arrives, directs the job to the sensor(s) with the best quality, capability, and availability. Data fusion would then be moved away from users into this centralized location.◆

References

1. Alberts, D.S., J.J. Garstka, and F.P. Stein. Network Centric Warfare. 2nd ed. CCRP Publication Series, Feb. 2000 <www.dodccrp.org/publications/pdf/Alberts_NCW.pdf>.
2. Federation of American Scientists Military Analysis Network. Washington, D.C. <http://fas.org/man>.
3. Garstka, John J. "Network Centric Warfare: An Overview of Emerging Theory." Phalanx, The Bulletin on Military Operations Research Dec. 2000.
4. AT&T Labs Research. "Graphviz - Open Source Graph Drawing Software." Vers. 1.12 Feb. 2004 <www.research.att.com/sw/tools/graphviz>.
5. Hall, D.L. Mathematical Techniques in Multisensor Data Fusion. Norwood, MA: Artech House, 1992.
6. Strömberg, D. "Integration och Styrning av Sensorer i Nätverk." ("Integration and Management of Sensors in Networks.") FOI, Totalförsvarets Forskningsinstitut (The Swedish Defence Research Agency), Linköping, Sweden, 2001.

Additional Reading

1. Schaar, D. "Network Centric Warfare: System Interoperability EKI 2003:02." Department of Management and Economics, Linköping Institute of Technology, Linköping, Sweden, 2003.
2. The Data Fusion Server <www.data-fusion.org>.

About the Author

David Schaar is a consultant with Booz Allen Hamilton in McLean, Va., where he is a cost analysis consultant with a primary emphasis on Command, Control, Communications, Computers, Intelligence, Surveillance, and Reconnaissance systems. This article was written during his previous capacity as a cost research analyst with PRICE Systems, LLC, of Mt. Laurel, N.J., and is based on research performed as part of his master's thesis. At PRICE Systems, Schaar developed an information technology affordability management tool for the company's activity-based costing solution. Schaar has a Master of Science in industrial engineering and management from the Linköping Institute of Technology in Sweden.

Booz Allen Hamilton
8283 Greensboro DR
B6037
McLean, VA 22102
Phone: (703) 377-8041
E-mail: schaar_david@bah.com

Letter to the Editor

Dear CrossTalk Editor,

Paul McMahon's article, "Bridging Agile and Traditional Development Methods: A Project Management Perspective" in the May 2004 edition of CrossTalk on bridging between agile and traditional development methods may have missed the real point. An on-site customer representative for a subcontractor in an environment where the customer is encouraged to change requirements can have serious risks, not only for the prime, but also for all of the other subs that have to adjust to those changes. Integration is far harder than straight development precisely because the communication cost of keeping the various pieces working together is large.

Often embracing change means never having to get it right. This has been a primary cause of failure on many so-called agile projects. (The most famous XP project was what should have been a routine payroll system at Chrysler that was cancelled prior to completion due to cost overruns and late deliveries of needed functionality.)

Good up-front architecture and good design mitigate the risks. Both the architecture and the implemented design need to allow for managed change. McMahon does this by adding process weight to agile methods in the form of his recommendations.

Actually, I believe that his modification to the waterfall model, or some other similar modifications, are pretty common to successful development regardless of whether any subcontractors are agile or not.

So, I would contend that the real point of McMahon's article is that successful development is not about adapting to XP by moving toward the middle. It is about the middle being in the right place in the first place because extremes in either direction create extreme risks. The XPers need to move toward the middle as well. If they ever want to build in a true system-of-systems environment, they will recognize that while change is itself a requirement, it needs to be accepted, managed, and controlled, but not embraced.

For a humorous, yet capable, description of the pitfalls (and positives as well) of XP, check out the book "XP Refactored," by Matt Stephens and Doug Rosenberg. It is a combination of clinical dissection and gossipy tell-all about XP. And the only thing extreme about it is the humor.

Gary A. Ham
Senior Research Scientist
Battelle Memorial Institute
(540) 288-5611 (office)
(703) 869-6241 (cell)

The opinions in this letter are the author's and do not represent Battelle Memorial Institute as a whole.

CrossTalk *invites readers to submit their thoughts, comments, and ideas on its themes and articles as a "Letter to the Editor." Simply e-mail letters to* <crosstalk.staff@hill.af.mil>.

Systems Engineering and Shoe Polish

I am writing this column on a flight from Albuquerque to Salt Lake City on route to the 2004 Systems and Software Technology Conference. Before we took off, the pilot triggered the microphone and made basically the following announcement: "Good morning … uh … this is your captain. We're on flight … uh, 1577 … from, uh, …" The speech seemed to go on forever, punctuated at frequent intervals by long pauses and "uhs." The person sitting in the seat next to me remarked, "I sure hope that he can fly the plane better than he can talk!"

I guess I am getting old and crotchety (no e-mail acknowledgements, please!). But I have noticed that when cockpit personnel key the mike to give us updates, sometimes it appears that they don't understand that speaking skills convey an impression. Having a competent airline pilot unable to form a complete sentence in less than 90 seconds of mike time – well, it makes me worry. (Before I get banned from future airline flights, I do understand that the pilot was probably completing a checklist and making the announcement at the same time. And the checklist was much more important. Still, the rambling announcement did little to inspire passenger confidence).

You've all heard the expression, "Put your best foot forward." Well, sometimes it helps to put a bit of shoe polish on the foot, too. Oftentimes, we forget that appearances really count. Many, many years ago, I had the opportunity to make a proposal for some consulting work. While I did the background material, one of my co-workers was responsible for the sales pitch itself. I had crunched the numbers and submitted accurate and up-to-date information for the sales pitch. Imagine my amazement to find out that the sales pitch was accomplished on hand-scribbled overheads. The company we were presenting our sales pitch to was also probably amazed to find the name of their company incorrectly capitalized! Needless to say, we were not overwhelmed by their desire to give us business!

I think that as engineers we sometimes forget that format is at least as important as content. As a software engineer, I often need to present information and documents to customers and users. I recently watched another engineer give a presentation using an out-of-focus projector that was partially blocked by his own laptop. After his 15-minute presentation was over, the next speaker moved and focused the projector – and many in the audience applauded. Nobody had really been listening to the presentation; they were concentrating on the annoyingly out-of-focus projector.

Documentation falls under the same umbrella. Tables of content, lists of figures, easy-to-use indexes – these all give you a good feeling about the content of the material. If it looks good then there is the perception that it also contains good information. While I am not suggesting for a moment that a glitzy color cover and fancy formatting will cover up poor quality material, I am suggesting that misspellings and sloppy formatting *will* cover up good quality material. Recently, I presented a report to a customer, and accompanying the report was a backup CD with data. I had labeled the CD with a felt-tip marker. My co-worker saw what I was doing, and without saying a word, went and printed a CD label with some simple artwork. It made a world of difference! The message was that I had taken care of the details – and it made the customer have more confidence in everything else!

And, last but not least – recognize that not every person has been given the ability to speak in front of an audience. I have worked on team presentations where the senior team member gave the briefing. Unfortunately, being the senior member didn't translate into speaking ability. A mumbling, stumbling monotonic report did little to impress the listeners; the important message our team was trying to convey was quickly lost due to lack of interest.

Impressions are quick to form, and hard to forget. Do not let good research and good work go ignored because of a lack of follow-through. No matter how you present your work to others, you want the presentation to convey this message, "This is high-quality work!"

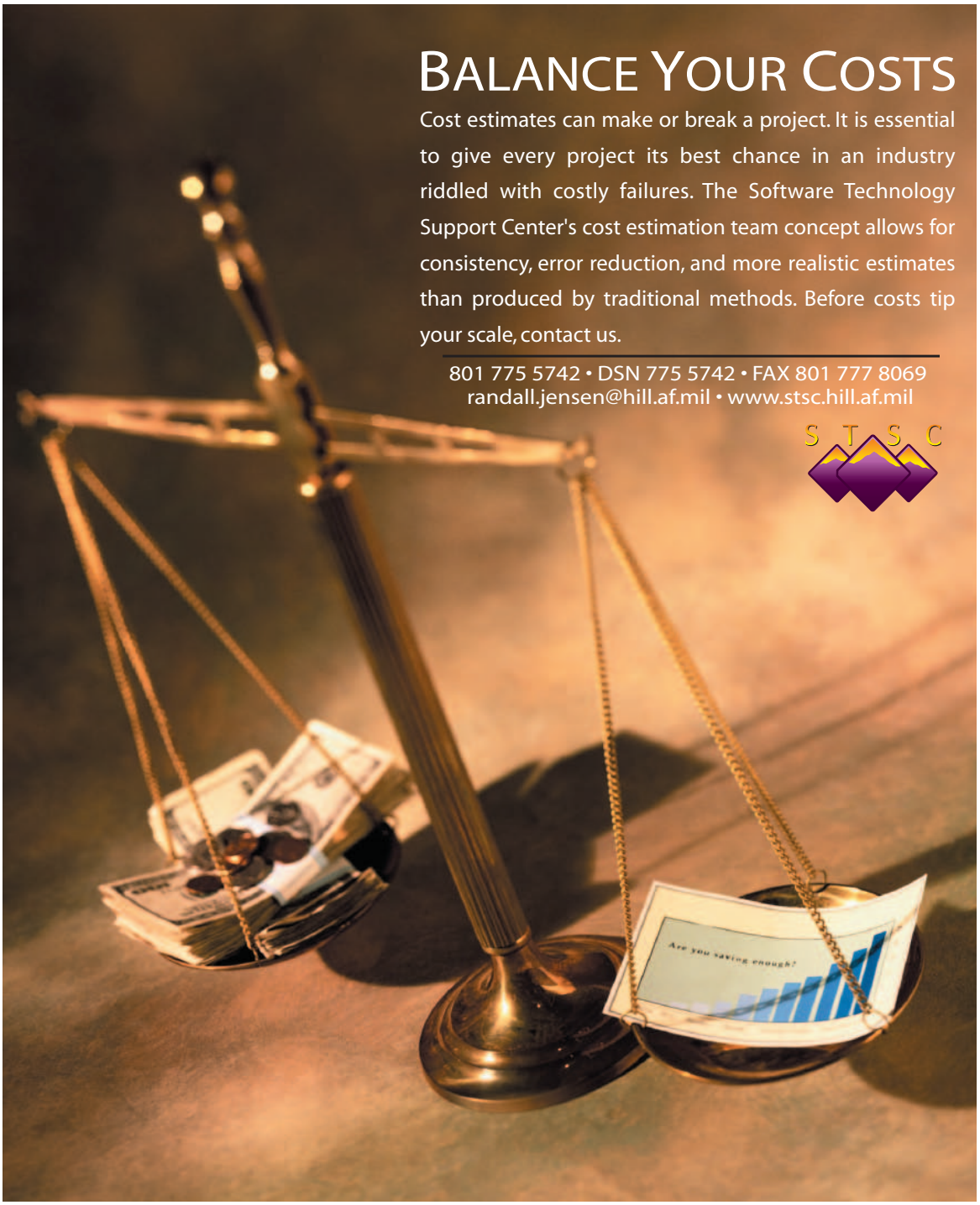
So, when you go to put your best foot forward, consider that a good coat of shoe polish will help make a good impression.

— David A. Cook
Senior Research Scientist
The AEgis Technologies Group, Inc.
dcook@aegistg.com

Can You BACKTALK?

Here is your chance to make your point, even if it is a bit tongue-in-cheek, without your boss censoring your writing. In addition to accepting articles that relate to software engineering for publication in CROSSTALK, we also accept articles for the BACKTALK column. BACKTALK articles should provide a concise, clever, humorous, and insightful perspective on the software engineering profession or industry or a portion of it. Your BACKTALK article should be entertaining and clever or original in concept, design, or delivery. The length should not exceed 750 words.

For a complete author's packet detailing how to submit your BACKTALK article, visit our Web site at <www.stsc.hill.af.mil>.



Balance Your Costs

Cost estimates can make or break a project. It is essential to give every project its best chance in an industry riddled with costly failures. The Software Technology Support Center's cost estimation team concept allows for consistency, error reduction, and more realistic estimates than produced by traditional methods. Before costs tip your scale, contact us.

801 775 5742 • DSN 775 5742 • FAX 801 777 8069
randall.jensen@hill.af.mil • www.stsc.hill.af.mil

S T S C

S T S C

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