

CROSSTALK

September 2007 *The Journal of Defense Software Engineering* Vol. 20 No. 9



SERVICE-ORIENTED ARCHITECTURE

Report Documentation Page				Form Approved OMB No. 0704-0188	
Public reporting burden for the collection of information is estimated to average 1 hour per response, including the time for reviewing instructions, searching existing data sources, gathering and maintaining the data needed, and completing and reviewing the collection of information. Send comments regarding this burden estimate or any other aspect of this collection of information, including suggestions for reducing this burden, to Washington Headquarters Services, Directorate for Information Operations and Reports, 1215 Jefferson Davis Highway, Suite 1204, Arlington VA 22202-4302. Respondents should be aware that notwithstanding any other provision of law, no person shall be subject to a penalty for failing to comply with a collection of information if it does not display a currently valid OMB control number.					
1. REPORT DATE SEP 2007		2. REPORT TYPE		3. DATES COVERED 00-00-2007 to 00-00-2007	
4. TITLE AND SUBTITLE CrossTalk: The Journal of Defense Software Engineering. Volume 20, Number 9, September 2007				5a. CONTRACT NUMBER	
				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) OO-ALC/MASE,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 Same as Report (SAR)	18. NUMBER OF PAGES 32	19a. NAME OF RESPONSIBLE PERSON
a. REPORT unclassified	b. ABSTRACT unclassified	c. THIS PAGE unclassified			

4 The Security of Web Services as Software

This article both describes the content of Special Publication 800-95 and highlights its critical omissions in terms of measures needed to produce Web service software that is in and of itself secure.

by Karen Mercedes Goertzel

10 Four Pillars of Service-Oriented Architecture

This article outlines four pillars to Service-Oriented Architecture (SOA) success and proposes how a Department of Defense organization can develop and implement an effective strategy for SOA implementation.

by Grace A. Lewis and Dr. Dennis B. Smith

14 Defining Services Using the Warfighter's Language

This article emphasizes the need to implement services in a way that reflects the warfighter's needs and expectations.

by Michael S. Russell

20 Applying a Service-Oriented Architecture to Operational Flight Program Development

This article describes how an SOA was successfully applied to re-use data and applications previously deployed in single-user, single-computer configurations.

by Mitch Chan

Open Forum

24 For Net-Centric Operations, the Future Is Federated

This article outlines several key strategies for leading the way to a true net-centric approach to SOA life-cycle quality and testing.

by John Michelsen



Cover Design by
Kent Bingham

Departments

3 From the Publisher

9 Coming Events Call for Articles Ad

16-17 SSTC 2007

30 Web Sites More Online From CROSS TALK

31 BACKTALK

CROSSTALK

CO-SPONSORS:

DoD-CIO The Honorable John Grimes

NAVAIR Jeff Schwalb

76 SMXG Kevin Stamey

309 SMXG Norman LeClair

402 SMXG Diane Suchan

DHS Joe Jarzombek

STAFF:

MANAGING DIRECTOR Brent Baxter

PUBLISHER Elizabeth Starrett

MANAGING EDITOR Kase Johnstun

ASSOCIATE EDITOR Chelene Fortier-Lozancich

ARTICLE COORDINATOR Nicole Kentta

PHONE (801) 775-5555

E-MAIL crosstalk.staff@hill.af.mil

CROSSTALK ONLINE www.stsc.hill.af.mil/crosstalk

CROSSTALK, The Journal of Defense Software Engineering is co-sponsored by the Department of Defense Chief Information Office (DoD-CIO); U.S. Navy (USN); U.S. Air Force (USAF); Defense Finance and Accounting Services (DFAS); and the U.S. Department of Homeland Security (DHS). DoD-CIO co-sponsor: Assistant Secretary of Defense (Networks and Information Integration). USN co-sponsor: Naval Air Systems Command. USAF co-sponsors: Oklahoma City-Air Logistics Center (ALC) 76 Software Maintenance Group (SMXG); Ogden-ALC 309 SMXG; and Warner Robins-ALC 402 SMXG. DHS co-sponsor: National Cyber Security Division of the Office of Infrastructure Protection.

The USAF Software Technology Support Center (STSC) is the publisher of CROSS TALK, providing both editorial oversight and technical review of the journal. CROSS TALK's mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.



Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail us or use the form on p. 13.

517 SMXS/MXDEA
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 CROSS TALK editorial board prior to publication. Please follow the Author Guidelines, available at www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf. CROSS TALK does not pay for submissions. Articles published in CROSS TALK remain the property of the authors and may be submitted to other publications.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CROSS TALK.

Trademarks and Endorsements: This Department of Defense (DoD) journal is an authorized publication for members of the DoD. Contents of CROSS TALK are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, the co-sponsors, or the STSC. All product names referenced in this issue are trademarks of their companies.

CrossTalk Online Services: See www.stsc.hill.af.mil/crosstalk, call (801) 777-0857 or e-mail stsc.web.master@hill.af.mil.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.



SOA Provides Opportunities and Challenges



Joe Jarzombek, the Department of Homeland Security (DHS) Director for Software Assurance, stated that with more organizations considering Service-Oriented Architectures (SOAs) for their applications and services and with software as a service gaining traction, we need to directly address the reality that these are built on software applications and also address factoring security needs into development, acquisition, and deployment decisions. SOAs provide an avenue to develop, operate, and access distributed computing systems. While traditional quality and information assurance methodologies tended to focus on closed or tightly-controlled monolithic systems, the reality of SOA has moved us away from that conventional thinking and forced the mainstream Information Technology establishment to operate in a complex distributed computing world.

SOAs provide numerous conveniences and additional challenges along with those conveniences. As the highlighted co-sponsor for this issue of CROSSTALK, the DHS focuses on the security challenges of the enabling applications for SOAs.

With this important focus, we start this month's CROSSTALK with *The Security of Web Services as Software* by Karen Mercedes Goertzel. In this article, Goertzel stresses the need to focus on security while developing the software that enables SOAs. We take a step back with the following article to focus on enablers of overall SOA success. Grace A. Lewis and Dr. Dennis B. Smith discuss developing an appropriate SOA strategy, implementing effective SOA governance, making sound technology assessments, and accounting for the fact that SOA requires a different mindset in *Four Pillars of Service-Oriented Architecture*. Next, Michael S. Russell addresses practical applications for the warfighter in *Defining Services Using the Warfighter's Language*. Mitch Chan provides an example of a current successful application in *Applying a Service-Oriented Architecture to Operational Flight Program Development*. In our final article, *For Net-Centric Operations, the Future Is Federated*, John Michelsen suggests a system intended to ease sharing of burden for shared software services.

Total SOA Assurance represents the merging of what was considered distinct domains of computer system assurance. Quality assurance, information assurance, and operations are all merging to a place where one cannot exist without the other. The buyers of technology and consulting services around SOA must become well-educated consumers. To that end, we should focus on current and developing concepts around total SOA assurance. As the nature of future distributed computing systems will cross organizational and political boundaries, a new framework of thinking about SOA needs to begin to handle the myriad of issues that surround interoperation organizations.

As the increasingly popular delivery scheme, SOA provides the enabling framework for software applications, yet it tends to further separate the user from the enabling software. Lest this separation cause the community to forget the challenges posed and faced by all involved with the software, the articles in this issue of CROSSTALK are intended to bring some key ideas to the forefront.

Elizabeth Starrett
Publisher



The Security of Web Services as Software

Karen Mercedes Goertzel
Booz Allen Hamilton

To help creators of Web services and Service-Oriented Architectures (SOAs) understand and address the security challenges that confront them, the National Institute of Standards and Technology (NIST) is getting ready to publish a new Special Publication (SP) 800-95, Guide to Secure Web Services. This SP describes Web service security standards and explains how to develop Web services and SOA portals using technologies based on those standards. However, neither SP 800-95 nor the standards it describes address a critical challenge: the security of Web services as software. Without considering software security, developers cannot create Web services that are truly trustworthy. This article describes both the content of SP 800-95 and highlights its critical omissions in terms of measures needed to produce Web service software that is in and of itself secure.

Web services in SOAs allow applications to interact and data to be interchanged without direct human intervention. The use by Web services of eXtensible Markup Language (XML), SOAP (formerly Simple Object Access Protocol), and related open standards enables these interchanges to be achieved over connections that are established dynamically on an *ad-hoc* basis.

Web service-based SOAs are proliferating exponentially, both in number and extent, in Department of Defense (DoD) and other government agencies, the private sector, and academia. Web services are relied on to create, manipulate, and protect information, including the most sensitive private information. They are used to perform high-consequence and mission-critical information-handling functions.

The individual, software-intensive Web services of which these SOAs are composed are also growing larger and more complex, to the extent that their own developers can no longer fully recognize – let alone comprehend – all of their possible behavioral states, weaknesses, and vulnerabilities.

At the same time, software-level threats to Web services and portals to SOAs are increasing in intensity, sophistication, and variety. Zero-day vulnerabilities in commercial Web service software products have not only been proven to exist, they are on the rise. (A zero-day vulnerability is one against which an exploit is launched by an attacker before a security patch can be issued by the product's developer.)

The security challenges presented by the Web service processing model, in which SOAP messages encapsulating sensitive XML documents are forwarded along complex chains of consumer,

provider, and intermediary Web services, are formidable. This is because many of the features that make Web services and SOAs attractive – greater accessibility of data, dynamic establishment of interservice relationships and communication paths, a high degree of service autonomy, and a minimal amount of direct human involvement – are all at odds with traditional security models and controls.

The nature of Web services and SOAs makes them subject to unique attacks, as well as variations (often involving intensification) of familiar attacks that already target Web servers and applications. Achieving secure Web service processing entails the extension and augmentation of existing Web and Internet security mechanisms. This is increasingly achieved through the implementation of authentication, authorization, trust, confidentiality, integrity, and availability of functions based on relatively new and still-emerging Web services security standards.

NIST SP 800-95, Guide to Secure Web Services

To help the architects, developers, and engineers involved in the creation of Web services and SOAs understand and address the security challenges that confront them, the NIST is publishing a new SP 800-95, *Guide to Secure Web Services*. The current draft of SP 800-95 can be downloaded from the SP page of the NIST Computer Security Resource Center Web site: <<http://csrc.nist.gov/publications/nistpubs/>>. The final version will be published at this address later this year.

The objective of this NIST SP is to describe and make sense of the broad range of Web service (WS) security stan-

dards that have been produced by the Organization for the Advancement of Structured Information Standards, World Wide Web Consortium, Web Services Interoperability Organization (WS-I), the Liberty Alliance, and other standards bodies. Because of this objective, the SP tends to focus on security issues that are already being addressed by these various standards bodies and omits discussion of those security challenges that have not yet been acknowledged by the standards community, or that the community has deemed *unimportant, out of scope, or too hard* to address via Web services security standards.

While SP 800-95 does describe how to implement security functions and protections for Web services and SOA portals, it does so almost exclusively from the point of view of what can be achieved using technologies based on current and emerging Web services security standards.

Unsurprisingly, after providing background information on key Web services and SOA concepts, capabilities, and components, including discovery, messaging, portals, coordination (choreography and orchestration), and the roles, modes, and properties of Web services, the SP's discussion is limited to what is already widely accepted as constituting *Web services security*: secure interservice messaging, protection of XML-based content, secure negotiation of contracts between service providers, and establishment of trust relationships between services.

For each of the security capabilities and protections it introduces, the SP describes the associated Web services security standards. These standards are organized into a stack (comparable to the Open Systems Interconnect [OSI] and Transport Control Protocol/

Internet Protocol network protocol stacks). Within an SOA, the standards are also frequently implemented within core security services, i.e., security services that other services depend on to perform essential functions on their behalf. The SP also describes the security threats to Web services that protections and functions based on these standards are intended to address. These functions and protections fall into eight general categories:

1. **Authentication and identity management.** The SP addresses service-to-service authentication and service, and identity management and authentication of human users who access SOAs via Web portals. Specific standards discussed include WS-Security for interservice authentication (including its security shortcomings), and Security Assertion Markup Language (SAML) as the basis for single sign-on by human users.
2. **Interservice trust.** The SP addresses trust establishment (based on authenticated identity) between services, and federation of trust across SOA boundaries.
3. **Secure service discovery.** The SP focuses on security for Universal Description, Discovery and Integration (UDDI) and Web Service Description Language (WSDL) interfaces, secure access to the SOA registry, secure interaction of Web portals with the SOA discovery services, and application programmatic interfaces (APIs) for secure service inquiry and publishing.
4. **Distributed authorization and access management.** The SP discusses the authorization of privileges to Web services and the enforcement of least privilege in SOA authorization models. Specific standards discussed include SAML as the basis for asserting privileges, eXtensible Access Control Markup Language (XACML) as the basis for service-level access control, and the use of XML schema and security metadata for data/content-level access control.
5. **Confidentiality and integrity of service-to-service interchanges.** The SP discusses use of HyperText Transport Protocol Secure (HTTPS) for transport layer security in SOAs, WS-Security for SOAP message-level security, XML encryption and signature for content protection, and the role of XML gateways in providing additional message-level and content-

level integrity protection.

6. **Accountability.** The SP discusses methods for achieving accountability end-to-end throughout a Web service chain, including audit in SOA environments and use of XML signatures as the basis for non-repudiation of Web service transactions.
7. **Availability and quality of service (QoS).** The SP discusses availability and QoS techniques such as fail-over, handling of service deadlocks and service recursion, and it addresses the security implications of competing reliable messaging standards.
8. **Security policy specification.** The SP discusses WS-Policy and its role in supporting specification and enforcement of security policy within an SOA.

Recognizing that standards are only the basis for implementations, the SP touches on tools and technologies for

“What Web service software does share with other software is its exposure to threats throughout its lifetime, not just after it has been deployed, but while it is still under development.”

implementing security standards-based Web services, such as Web service-oriented developer toolkits, XML parsers, procedural languages commonly used in Web service development, XML, and tools and techniques for testing the security of Web services.

In this context, the SP also addresses issues associated with secure Web service enabling of legacy applications (e.g., public-key enabling consistent with Web services standards and implementing standards-based security functions and protections for legacy applications and databases exposed as Web services).

Security of Web Services as Software

Though SP 800-95 does cover implementation of Web services that are likely to be called secure because they are

based on accepted Web services security standards, the SP does not discuss what is probably the most important security issue in the implementation of Web services – an issue that is not addressed by any Web services security standard: The security of Web services as software.

It can be argued that because Web services are subject to the same security issues as all software, and particularly network-based application software, no discussion of the topic was needed in NIST’s Web services security guidance. Indeed, NIST excluded such a discussion as out of scope exactly because the need for software security is not unique to Web services.

This is an interesting position for NIST to take, given that one can argue equally that requirements for confidentiality, integrity, availability, authentication, trust, identity management, etc., are not unique to Web services either. Moreover, in several cases the standards that are being used to address these requirements were never intended to be exclusive to Web services. For example, XML digital signature and encryption are intended for use with all XML documents, not just those exchanged between Web services. SAML and XACML, while certainly used to implement authentication and access control in Web service implementations, are intended to be widely applicable to all types of applications.

Threats to Web Service Software

What Web service software does share with other software is its exposure to threats throughout its lifetime, not just after it has been deployed, but also while it is under development. Threats to software in development may be intentional, coming from malicious developers who subvert or sabotage the software they or their colleagues build, or they may be unintentional, introduced by developers who, due to ignorance, carelessness, or pressure to get the software out on time fail to implement checks and controls that would eliminate or minimize exposure of the software’s numerous weaknesses and vulnerabilities.

Compounding this problem is the almost universal use of commercial and open source software components in the building of Web services. The pedigree of such components is often a mystery. Neither the processes used to develop those components, nor the security properties, weaknesses, and vulnerabilities of the components themselves are ever investigated by the developers who

incorporate those components into their Web services. Even pedigree of open source components is taken entirely on faith. Just because open source code is available for review does not mean that anyone actually does code reviews of open source code.

Specific threats to Web services under development fall into two general categories: 1) malicious code, and 2) exploitation of weaknesses and vulnerabilities.

1. Malicious code. Logic bombs, time bombs, Trojan horses, worms, and other undocumented malicious functions are intentionally inserted into the source code or appended to the binary executable by the developer. Failure to review source code and carefully observe the behavior of executing binary code means that such embedded malicious code will be allowed to remain in software when it is deployed. Furthermore, malicious code may be added by an attacker who intercepts and tampers with electronically distributed executables. After the software is deployed, it is subject to the delivery of new malicious code or the execution of embedded malicious code. The risk of malicious code insertions and executions is increased when the software and its environment are not configured insecurely, and anti-malicious code countermeasures are not deployed or used effectively.

2. Exploitable weaknesses and vulnerabilities. Flaws, defects, errors, and faults are often included, usually unintentionally but sometimes intentionally, in the artifacts – specification, architecture, design, or implementation – of the Web service. In some cases, these can be intentionally leveraged by attackers (or by malicious software processes acting on an attacker's behalf) to compromise the security of the Web service itself or of the data it handles.

Weaknesses originate as early as the requirements phase. Security-related requirements may be overlooked or misstated, or spurious requirements may be included. They may arise as the result of poor architecture or design choices, such as failure to enforce least privilege or to design in redundancy of critical processes.

Vulnerabilities may enter software during its implementation, due to use of non-secure implementation practices. Examples of such practices include: accepting user input without

first validating it; using vulnerable technologies (e.g., SOAP over unencrypted, unauthenticated HTTP connections); use of non-secure programming languages (e.g., non-type-safe languages used without input validation) and library functions (e.g., buffer overflow-prone C functions such as `printf`); use of non-secure development tools (e.g., compilers that do not perform bounds checking); reuse of vulnerable components (e.g., commercial software that has known vulnerabilities); use of development tools (e.g., compilers that do not perform bounds checking); or reuse of vulnerable components (e.g., commercial software that has known vulnerabilities).

Weaknesses and vulnerabilities may be allowed to remain in software due to the failure to perform adequate security reviews, assessments, and tests of the artifacts of the development process (from specifications through the software itself); or the intentional tampering with the results of such reviews/assessments/tests.

They may also be allowed to remain in the form of back doors and trapdoors that are not removed prior to software distribution. They may arise or fail to be mitigated due to specification of non-secure configuration parameters for the software and its environment (or the use of non-secure installation procedures, scripts, and tools).

Once the Web service is operational, it is subjected to misuse by its intended users, and abuse by attackers. Understanding the attack patterns to which Web services are likely to be subject can be extremely helpful to the developer in specifying security requirements, architectural characteristics, and design properties that can reduce a service's exposure and vulnerability to likely attack patterns. Moreover, such understanding provides a basis for defining the code assessment criteria and security test plans for developmental and non-developmental Web service software components (i.e., attack surface definitions highlight specific targeted security flaws to look for during code reviews; misuse and abuse cases can be elaborated into white-box and black-box test scenarios).

Exploits Against Web Services

The exploits, or attacks, that target existing Web services fall into two main categories: direct and indirect. Direct attacks

exploit known or suspected vulnerabilities and weaknesses in the Web service itself, while indirect attacks may target the service's interface with the environment and middleware components on which it relies, or its interface with the external services and applications with which it interacts.

Attacks against Web services have one of three general objectives:

- 1. Disclosure.** This may be achieved through reconnaissance attacks that discover or reveal Web service vulnerabilities that can be exploited by other attack patterns. It may also be accomplished through attacks that bypass or cause denial of service in the Web service so as to directly access and disclose the sensitive/private data handled by that service.
- 2. Subversion.** This includes subversion of the service's functionality (i.e., by direct tampering, malicious code insertion/delivery, command injection, tampering with the state of the service's execution environment, and intentional triggering of errors or faults at the service boundary with its environment), of its data, or of its security assumptions about other services (i.e., by an attacker or malicious process masquerading as an entity or hijacking an entity that is trusted by the service, and thereby escalating its own privileges to match those of the trusted entity). It also includes attacks that bypass or cause denial of service in the service in order to directly access or tamper with the data the service handles.
- 3. Sabotage.** This may be achieved through denial-of-service attacks on the service itself, or on the external entities on which it depends for its dependable, secure operation (e.g., execution environment and network components, core security services, and defense-in-depth protections). An objective of sabotage is often to bring down or bypass the targeted software in order to directly access the data in control.

Particular Security Challenges for Web Services in SOAs

Some security challenges are unique to Web service software, and others are greatly exacerbated when they arise in Web service software. The chains of dependencies between autonomous, dynamically invoked Web services within SOAs are often much more complex than when autonomous software components are used in more traditionally

distributed processing models. In most distributed systems, the list of components that will be invoked, and the order in which they will be invoked, in the course of completing a particular transaction or task is predefined, as to a great extent are the outputs of the invoked components. In an SOA in which services are dynamically coordinated (through choreography or orchestration), it is frequently impossible to predict in advance which services will be invoked by other services, and in what order those invocations will take place. In dynamic coordinations that cross the boundaries from one trust domain to invoke services in another trust domain, it is especially hard to establish valid security assumptions in advance about the behaviors, policies, and permissions expected by the services in the remote domain. If a fault in a Web service causes that service to violate expected behavior or policy, the results of such a violation have the potential to propagate throughout the entire chain of services. Because that chain is unpredictable (being dynamically established rather than pre-defined), the propagation and impact of the violation will also be unpredictable. The result of a fault in one Web service, then, may compromise the security and dependability of other Web services much further along the chain, which may make forensic analysis to identify the true source of the compromise and to trace all the possible branches of its progress extremely difficult (if not impossible).

Moreover, in SOA implementations, each service is inherently dependent on other autonomous services. The increasingly widespread use of what are termed *core security services* model means that many innately non-secure Web services depend on other services for critical security protections and capabilities. Their role as security service providers means that these core services are not only the most critical services in the SOA, but represent the highest-value targets to attackers.

Even when provided via core security services, the SOA's security functions and protections are often actually implemented using the security functions and protections provided by the underlying application framework, e.g., Java Enterprise Edition or .NET. This increases the risk that consumer Web services not based on the same framework technologies may not interoperate seamlessly with the core security services. The centralization of the SOA's security func-

tions into a set of core services increases the imperative to ensure that such services, which will be trusted to guarantee the entire SOA's security posture, will be able to resist or tolerate attacks and to continue operating reliably under hostile conditions. If such software contains weaknesses and vulnerabilities that can be exploited by attackers, this model collapses due to the misplacement of trust in components that are too vulnerable to perform their designated tasks.

A number of other factors unique to Web services and SOAs make their software components more vulnerable to software-level exploits than other types of application software. First and foremost among these are the following:

1. The woefully inaccurate assumption that Web service interfaces will be used only as intended by other Web services, and not by human beings or

“Understanding the attack patterns to which Web services are likely to be subject can be extremely helpful to the developer in specifying security requirements, architectural characteristics, and design properties that can reduce a service’s exposure ... ”

malicious processes. Because Web services generally have no direct human interfaces, their operation receives little if any human scrutiny or intervention ... *except* by attackers.

2. The unavoidable fact that by exposing Web services applications and databases that were never originally intended for direct public access, the weaknesses and vulnerabilities of those applications and databases are also exposed to public view. Moreover, the easy-to-use development tools used by many Web ser-

vices developers obscure the services’ low-level functionality from the developer; it is these low-level functions that often contain the exploitable weaknesses and vulnerabilities that are targeted by attackers and malicious code. For example, Apache Axis 2 enables a Java developer to simply load his/her Java objects into the Axis SOAP engine. At runtime, it is the SOAP engine that determines which incoming SOAP request messages should be routed to which Java objects. The SOAP engine then translates those requests into standard Java function calls and routes them appropriately. Unless he/she has expressly reviewed the source code of the Axis SOAP engine, he/she will have no idea whether its routing or translation functions contain embedded malicious logic that could result in incorrect routing of messages or incorrectly generated Java calls. In the case of a commercial tool, such as Visual Studio, the ability to review the tool’s source code is not even an option.

Automatic discovery of Web services in particular is a feature of SOAs that makes it easier for attackers to locate and access potential targets. Publishing of repository entries about services through standard discovery interfaces (WSDL and UDDI) represents an unprecedented level of public disclosure of service processing details – details that can be used by reconnaissance attackers to craft much more effective attacks on the discovered services. Moreover, to accomplish automatic service discovery, privileges must be granted to unknown entities outside the organization that owns the services being discovered. There is a significant question as to the extent that entities outside the SOA, which interact with services discovered inside the SOA, can be governed by the policies (including security policies) enforced for that SOA.

For example, Organization #1, which operates SOA #1, may mandate that all Web service software must undergo code review and penetration testing before being deployed. Organization #2, which operates SOA #2, may have no such policy. So, if SOA #1 establishes a federated trust relationship with SOA #2, there is no way for Organization #1 to know whether the Web services in SOA #2 contain exploitable faults or malicious logic. Trust, as it is defined for Web services, (e.g., WS-Trust) refers solely to the assurance that a given service’s identity has been authenticated by a trusted third

party. By this definition, trust has nothing to do with the authenticated service's *trustworthiness*. The non-malicious functioning and behavior of the service must be taken entirely on faith.

Finally, all of the security problems associated with component-based software systems are also present in service choreographies and orchestrations, and furthermore, exacerbated by the increasingly *dynamic* nature of service composability. While security assumptions about individual services may be derived from the services' WSDL descriptions, and when services are combined in ways that differ from transaction to transaction, it is virtually impossible to establish (1) whether the security assumptions about a given service are still valid and meaningful when that service is instantiated within a given choreography/orchestration, and (2) whether there are any irresolvable security conflicts between services that are dynamically composed into a choreography or orchestration.

It is true that some features of Web service technology actually help mitigate security issues found in other types of software. Most notably, the reliance of Web services on platform-independent, standards-based APIs such as WSDL and SOAP, rather than using proprietary and/or platform-specific APIs, makes it easier to replace vulnerable Web services quickly with less vulnerable substitutes. Use of standard APIs also enables diversity of service implementations – a secure design principle that, when coupled with redundancy of services, reduces risk by reducing the number of services that will be compromised by an attack pattern tailored to exploit a specific vulnerability in a particular Web service implementation (or product). The result is an improvement in availability because the alternate services are unlikely to be susceptible to the same implementation-specific attack patterns that compromised the services they back up.

Building Secure Web Service Software

What can be done to make Web service software trustworthy? In practical terms, trustworthiness will be achieved by producing a Web service that is dependable, not only under both expected operating conditions but also under unexpected and intentionally hostile operating conditions. It is this dependability under unexpected hostile conditions that constitutes software *security*.

In practical terms, intentionally hostile operating conditions are created either by the presence of attack patterns or the behaviors that result from execution of malicious code. To continue operating dependably, then, a Web service must be designed and implemented so that it is able to do the following:

- Recognize and *resist* or block most attack patterns and malicious behaviors.
- *Tolerate* and safely handle the errors and failures that result from those attacks and malicious behaviors it cannot resist or block.
- Exhibit resilience by isolating and constraining the damage and recovering quickly (to an acceptable level of capability) from successful attacks and malicious behaviors.

“... intentionally hostile operating conditions are created either by the presence of attack patterns or the behaviors that result from execution of malicious code.”

Furthermore, to be deemed trustworthy, Web service software must not only exhibit the properties that constitute dependability and security, but also those that constitute *assurability*, which is the ability to independently verify the software's other required properties. The properties that constitute software dependability are correctness and predictable execution (i.e., the software does what it is supposed to do and nothing else), and in some cases safety. (Software safety is defined in the National Aeronautics and Space Administration Software Assurance Glossary <<http://sw-assurance.gsfc.nasa.gov/help/glossary.php>> as the systematic identification, analysis, tracking, mitigation, and control of software hazards and hazardous functions, hazards being existing or potential conditions or functions that can contribute to or result in mishaps or accidents. Software safety does not concern itself with preventing intentionally induced incidents, even though such an incident could result in a mishap or accident.)

The properties that constitute soft-

ware security are *integrity* (inability to subvert) and *availability* (inability to sabotage), and for Web services, *accountability* of the service as a non-human actor in a SOA (which includes *non-repudiation* by the service of its actions). In many cases, confidentiality of the software itself is also a desirable security property: The software's executable and/or operational behaviors may be hidden and/or obfuscated to make reconnaissance and disclosure of vulnerabilities difficult.

The properties that promote *assurability* include the following: *simplicity* (of design and implementation), *smallness* (of code), and *traceability* (of implementation to requirements).

In practical terms, a Web service can be said to be secure when it achieves the following:

1. The behavior of the service itself (including its behavioral state changes in response to inputs and external events) does not make the service vulnerable to attack or malicious code insertion/execution. This means the service must handle all inputs safely, validating them before use and rejecting or modifying (to make acceptable) those that threaten its secure behavior. It also means that the service must handle errors, exceptions, faults, and failures safely and securely, so that these events do not cause the software to enter an insecure state or compromise the data and resources to which it has access.
2. The service's interactions and interfaces must be secure. This includes those used among the service software's constituent components, e.g., remote procedure calls, and those used between the service and any external entities, including other Web services (e.g., SOAP over HTTPS with WS-Security), environment-level components (i.e., APIs, call-level interfaces), and human users (i.e., user interfaces).
3. The service is executable and data files in the file system must be protected from unauthorized access. This means that the configuration parameters of the service itself and of its execution environment protections must be as restrictive as possible.
4. The service's attack surface must be minimized. If this has not been achieved by reducing the number of vulnerabilities in the service itself (both at the architectural and software levels), it might be achievable through defense-in-depth measures that minimize the exposure of the vulnerabilities that were not eliminated.

Conclusion

While NIST's new SP 800-95, *Guide to Secure Web Services*, should prove helpful in increasing Web service implementers' knowledge and understanding of the security standards being adopted to secure service-to-service interactions within distributed SOA-based information systems, the SP does not discuss methods and techniques for design and implementation of secure Web service *software*. This leaves it up to the Web service developer to find that type of information elsewhere.

A good place for developers to start looking is the Department of Homeland Security's (DHS) BuildSecurityIn Web portal at <<https://buildsecurityin.us-cert.gov/>>. The resources here provide a broad range of recommendations on how Web service developers can add security principles and practices to their existing software processes so that the software produced by those processes will not only perform its required security functions, but will exhibit the levels of attack-resistance, attack-tolerance, and attack-resilience required to minimize its attack surface and susceptibility to malicious code penetrations and executions.

Recently, the Defense Information Systems Agency (DISA) published a Security Technical Implementation Guide entitled Application Security and Development Security. This can be downloaded at <<http://iase.disa.mil/stigs/stig/asd-stig.pdf>>. ♦

About the Author



Karen Mercedes Goertzel is a software security subject-matter expert supporting the Director of the DHS Software Assurance

Program and has provided similar support to the DoD's Software Assurance Tiger Team. From 2002-2004 she was project manager of the DISA Application Security Support Task and is currently leading the team developing NIST Special Publication 800-95, *Guide to Secure Web Services*. In addition to software assurance and application security, Goertzel has extensive expertise and experience in trusted systems and cross-domain information sharing solutions and architectures, information assurance (IA) and cybersecurity architecture, strategy and planning, risk management, and mission assurance. She has written and spoken extensively on software security and IA topics, both in the U.S. and abroad.

Booz Allen Hamilton
8283 Greensboro DR H5061
McLean, VA 22102
Phone: (703) 902-6981
Fax: (703) 902-3537
E-mail: goertzel_karen@bah.com

COMING EVENTS

October 2-4

STPCON 2007 Software Test and Performance Conference
 Boston, MA
www.stpcon.com

October 16-17

ICSQ '07 International Conference on Software Quality
 Denver-Lakewood, CO
www.asq.org/conferences/software-icsq-2007/index.html

October 22-25

10th Annual Systems Engineering Conference
 San Diego, CA
www.ndia.org

October 22-26



STARWEST 2007
Software Testing Analysis and Review
 Anaheim, CA
www.sqe.com/StarWest/

October 29-30

VERIFY 2007
 Crystal City, VA
<http://verifyconference.com>

November 4-7

AYE 2007
Amplifying Your Effectiveness
 Phoenix, AZ
www.ayeconference.com

May 2008



Systems and Software Technology Conference
www.sstc-online.org

COMING EVENTS: Please submit coming events that are of interest to our readers at least 90 days before registration. E-mail announcements to: nicole.kentta@hill.af.mil.

CALL FOR ARTICLES

If your experience or research has produced information that could be useful to others, CROSSTALK can get the word out. We are specifically looking for articles on software-related topics to supplement upcoming theme issues. Below is the submittal schedule for three areas of emphasis we are looking for:



Project Tracking

April 2008

Submission Deadline: November 16, 2007

Software Safety

May 2008

Submission Deadline: December 6, 2007

Information Assurance

June 2008

Submission Deadline: January 18, 2008

Please follow the Author Guidelines for CROSSTALK, available on the Internet at <www.stsc.hill.af.mil/crosstalk>. We accept article submissions on all software-related topics at any time, along with Letters to the Editor and BACKTALK. Also, we now provide a link to each monthly theme, giving greater detail on the types of articles we're looking for <www.stsc.hill.af.mil/crosstalk/theme.html>.

Four Pillars of Service-Oriented Architecture

Grace A. Lewis and Dr. Dennis B. Smith
Carnegie Mellon University, Software Engineering Institute

Among current technologies, Service-Oriented Architecture (SOA) has the greatest potential for implementing the vision of migration to net-centric operations. While SOA has been successful in many cases, it has also been marked by a number of expensive failures. This article outlines four pillars to SOA success that include the following: developing an appropriate SOA strategy, implementing effective SOA governance, making sound technology assessments, and accounting for the fact that SOA requires a different mindset. As a result, the article proposes how a Department of Defense (DoD) organization can develop and implement an effective strategy for SOA implementation.

A cornerstone of DoD policy for future software and systems policy is the migration of systems to net-centric operations. The net-centric vision requires the leveraging of a highly flexible set of capabilities that can be composed quickly and flexibly into applications that take advantage of the interoperable aspects of the web and provide effective mission value. Among current technologies, SOA has the greatest potential for implementing this vision.

However, there is a great deal of confusion about what SOA is, whether it is real, and what it takes to implement a SOA-based system. This article provides a high-level introduction to SOA, and then outlines how a DoD organization can develop an effective strategy for implementing the vision.

Basic SOA Concepts

SOA has become an increasingly popular mechanism for achieving interoperability between systems. It is a way of designing systems composed of services that are invoked in a standard way. Common goals for the adoption of SOA are to eliminate redundancy, assemble new functionality from existing services, adapt systems to changing needs, and leverage legacy investments. An SOA-based system is composed of the following:

- **Services:** These are reusable components that represent business or mission tasks, such as customer lookup, weather, sensor placement, account lookup, or credit card validation. Services can be globally distributed across organizations and reconfigured to support new tasks or missions. They are reusable because they can be utilized by many business processes or mission threads. They usually provide coarse-grained functionality, such as customer lookup as opposed to finer-grained functionality such as customer address lookup.
- **Service consumers:** These are clients

for the functionality provided by the services, such as end-user applications, systems, or even other services.

- **SOA infrastructure:** The infrastructure connects service consumers to services. It usually implements a loosely coupled, synchronous or asynchronous, message-based communication model, but other mechanisms are possible. The infrastructure often contains elements to support service discovery, security, and other operations. A common SOA infrastructure is an Enterprise Service Bus (ESB) to support Web Service environments. The Army's System of Systems Common Operating Environment and Defense Information Systems Agency's Net-Centric Enterprise Services are two examples of SOA infrastructures within DoD.

The benefits of SOA can be significant. However, SOA implementation is a complex engineering task and requires careful attention to engineering issues as well as to the four pillars for SOA success that are presented in this article.

Pillars for Successful SOA-Based Systems Development

It is common to view SOA-based systems development as a technical problem with a technical solution. However, successful SOA-based systems development requires attention to four pillars as illustrated in Figure 1:

- Alignment with mission and business goals.
- Instantiation of principles of SOA governance.
- Evaluation of relevant technologies for SOA implementation.
- Recognition that SOA requires a different mindset than traditional development.

Strategic Alignment

The first pillar, *Strategic Alignment*, focuses SOA decision-making on mission and

business priorities rather than the availability of vendor products, or preferences of individuals down the chain of command. If the wrong strategy is selected, it can result in an expensive collection of random services that are never used. A successful SOA strategy includes the following:

- Evidence of fulfillment of critical business goals.
- Alignment with organizational enterprise architecture and current and future Information Technology (IT) infrastructure.
- Realistic choices of technologies and infrastructures.
- Realistic and gradual adoption strategy.
- Adequate SOA governance structure.
- Priorities for implementation.
- Reuse strategy across internal and external organizations.

These issues can be addressed through activities that provide a focus to the SOA implementation, the overall business plan, identification of high priority business processes, and disciplined SOA adoption.

Focus to SOA Implementation

The high-level mission and business goals need to dictate the focus of an SOA implementation. As an example, four different high-level goals can lead to four different SOA strategies:

- An SOA-based system to support a battlefield will have critical needs to ensure performance, availability, and security.
- Increasing information available to stakeholders will focus on intuitive portals and creation of services related to information that is important to stakeholders.
- Integrating new partners will focus on a flexible SOA infrastructure, a very well-described service repository, and clear guidelines for composition.
- Maximizing security may lead to a proprietary SOA infrastructure.

Overall Business Plan

At a high level, there is recognition that SOA can provide agility, adaptability, legacy leverage, and integration with business partners. Current work has identified the business value of SOA for E-Commerce [1], E-Services, banking, and on-line services. In order to determine the amount of investment required and the projected payoff, an economic analysis needs to be planned at the beginning of an SOA implementation to identify the following:

- What constitutes a success within the context of a specific SOA implementation?
- How is return on investment measured?
- What are the resulting savings of SOA implementation (e.g. infrastructure consolidation, server and application virtualization, reuse of services, business agility)?

Identification of High Priority Business Processes

Any organization has many potential business process tasks that are candidates for services. Services are identified through a top-down analysis of business and mission processes, a bottom-up legacy system inventory, or a combination of the two. High-priority services are selected based on their relationship to critical goals. Traditional business process modeling, business process analysis, and business process reengineering techniques can help to model business processes and identify areas where services may be valuable. Although these methods will not model services, they suggest a starting point for setting priorities. Some of these approaches include the following:

- Enterprise architecture – analyzes business goals, what the business does, the type of information needed, and how the business uses IT to meet its goals.
- Business process analysis – models the business and its relationship to the external environment. This is an approach for identifying business processes that are candidates to become services.
- Business process modeling – analyzes and optimizes business processes to optimize current performance. This can provide details on the modeling of specific processes once they have been identified as candidates.
- Business process reengineering – analyzes current business processes and changes these processes, often in a radical way, to meet new business needs.

Disciplined SOA Adoption

An SOA implementation can start with a *big bang* approach that attempts to get SOA implemented at once throughout an enterprise. However, it is more prudent to begin with a pilot project that will provide a proof of concept. Pilot projects should focus on areas that provide high impact and visibility with the lowest risk. Gradual implementation can then lead to other projects that integrate a single organizational unit, to projects that integrate multiple business units, and later to large scale efforts that provide a virtual enterprise where all applications are built based on services [2].

SOA Governance

Governance has been rated as the main inhibitor of SOA adoption [3]. SOA governance provides a set of policies, rules, and enforcement mechanisms for developing, using, and evolving SOA-based systems, and for analysis of their business value. SOA governance includes policies, procedures, roles, and responsibilities for design-time governance and runtime governance.

Design-time governance includes elements such as rules for strategic identification of services, development, and deployment of services, reuse, and legacy system migration to services. It also enforces consistency in use of standards, SOA infrastructure, and processes.

Runtime governance develops and enforces rules to ensure that services are executed only in ways that are legal. Runtime governance procedures address concerns such as 1) access to applications and data, 2) the replacement of services, and 3) consistent interactions with the SOA

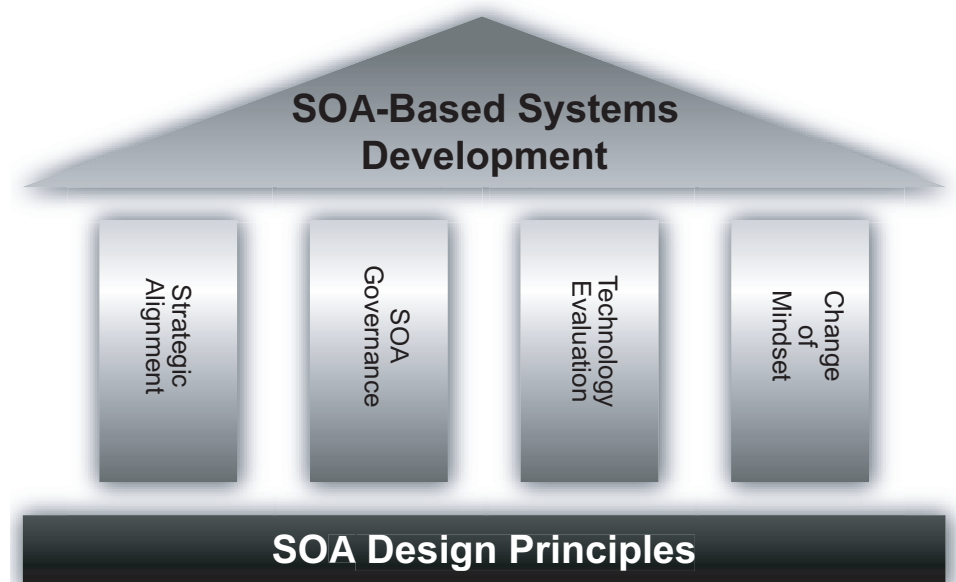
infrastructure.

Service-level agreements (SLAs) also fall under runtime governance. SLAs can include runtime validation of contractual specifications on performance, throughput, and availability; the use of automated metrics for tracking and reporting; and problem management.

A well-defined governance model needs to answer such questions as the following:

- What is the process for determining what services to create?
- What is the process for evolving and changing services if there are many consumers of the service?
- Many services can be common across several lines of business in an enterprise. Who *owns* these common services?
- Who owns the actual data if more than one service is using it?
- What is the resolution mechanism if there are conflicting requirements or change requests for shared services?
- What happens if the same (or similar) service is being developed by more than one service provider?
- What mechanisms, tools and policies are used for maintaining and monitoring deployed services?
- How are enterprise-wide policies enforced across various services both internally as well as externally to the organization?
- Who owns and maintains the shared repository of services in an organization?
- How are SLAs defined and enforced between service consumers and providers?

Figure 1: Pillars of SOA-Based Systems Development



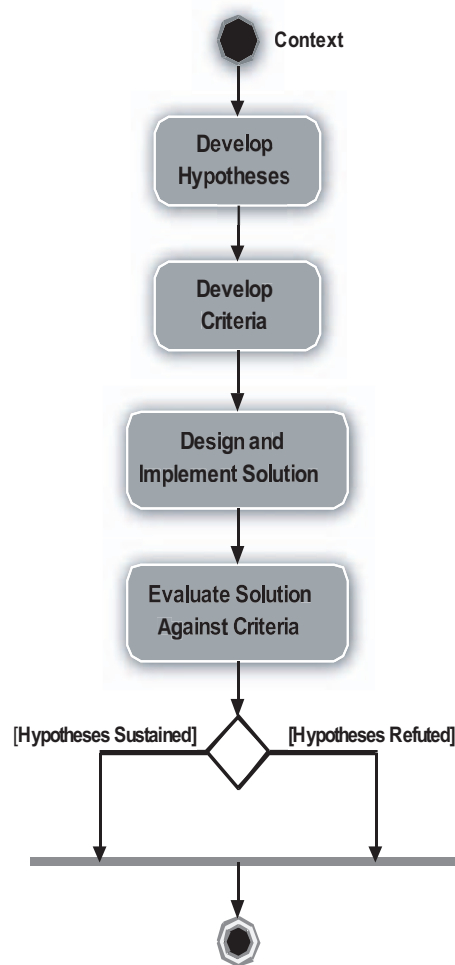


Figure 2: *T-Checks Approach for Technology Evaluation*

Technology Evaluation

Because an SOA implementation may use a number of technologies in novel contexts, it is important to evaluate whether a specific set of technologies is appropriate for the task at hand. Pillar 3 requires determining the fitness of a technology within a specific context before making a long term commitment to it. In adopting an SOA-based systems approach, a number of different technologies, standards and tools may be part of an implementation. Examples of these different technologies can involve specific web service standards, versions and tool implementations, cus-

tom infrastructures, ESBs, interfaces to specific databases, and language bindings.

It is easy to draw a slide showing how the pieces can fit together at an abstract level. However, all technologies work well within a specific context and under certain conditions. For example, Web services work well for asynchronous communication over the Internet. In a business environment these conditions are very common, but in a military tactical command and control environment this might not be the case because of high performance and availability requirements.

One way to perform this type of analysis is through a light weight evaluation method such as T-Checks [4, 5]. Other approaches can be used; however, the approach should enable a hands-on contextual analysis. The T-Checks approach is illustrated in Figure 2 and can be summarized in terms of the following steps:

- Identify technology requirements and context. Determine and document why the organization wishes to conduct the evaluation, what the expectations and concerns are with respect to the technology capabilities, and what is the context in which the technology plans on being used. Determine the environment in which the evaluation will take place, including expectations and constraints of the technology and measures of success.
- Develop hypotheses that are derived from the expectations placed on the technology. Hypotheses are claims about the technologies that are to be sustained or refuted.
- Develop criteria to determine if the results sustain or refute a hypothesis. Criteria are stated as a clearly measurable statement of capability. Each hypothesis can have one or more criteria, depending on the breadth covered by the hypothesis.
- Design and implement the experimental solution which is the simplest set of applications and/or components that are able to answer the questions posed

by the hypotheses and associated criteria, within a given scenario. The experimental solution is implemented, run, and observed, until there is enough information to sustain or refute the set of hypotheses.

- Evaluate the solution against criteria in order to make a decision with respect to the fitness of the technology for the context in which it is intended to be used. Based on the results of the evaluation there should be enough information to decide if it is the following:
 - A good fit with requirements.
 - Not a good fit with requirements.
 - Has some mismatches that could potentially be solved by modifying the context or modifying the technology itself.

Awareness of a Different Mindset

There are a unique set of challenges in building SOA-based systems. These challenges require a different development approach that deals with the characteristics of SOA-based systems. Although it is difficult to generalize, some of the contrasts of SOA systems versus traditional systems are presented in Table 1.

These differences impact the way software is developed throughout the life cycle:

- During requirements, it is important to have close ties to business process modeling and analysis. In addition, there is the need to anticipate potential service requirements from unknown consumers.
- During architecture and design, it is important to have technology evaluations and to perform explicit trade-off analyses.
- Implementation decisions will be impacted by emerging standards and may require simulation of the deployment environment.
- Testing requires a strong emphasis on exception handling and requires all test instances of services are available.
- Maintenance requires more sophisticated impact analyses and greater coordination of release cycles.

Because SOA implementation requires a different mindset than traditional software development and acquisition, it is important to develop an overall transition strategy to address how to acquire the new skills that may be required through training personnel, hiring new staff, or bringing in external experts. In addition SOA merges the technical and business worlds; therefore, it is important to have expertise

Table 1: *Some Differences Between Traditional Systems Development and SOA-Based Systems Development*

Traditional Systems Development	SOA-Based Systems Development
Tight coupling between system components	Loose coupling between applications and services
Shared semantics at design time	Semantics ideally enable dynamic discovery and execution of services
Known set of users and usage patterns	Potentially unknown service users and usage patterns
System components all within the same organization	Multiple organizations providing and supporting system components

in both fields. The fact that SOA implementations have the potential of crossing the enterprise also suggests the need for developing a perspective that spans the concerns of the entire enterprise, rather than just the issues of a specific program or business unit. As discussed in the section Disciplined SOA Adoption, a gradual adoption process that starts with small scale pilots and expands gradually is also recommended.

Conclusions

The SOA approach offers real value for DoD organizations as a technology for migrating toward net-centric operations. However, the rhetoric surrounding SOA can often be confusing and misleading. Establishing an effective SOA approach is a complex acquisition, management and technical task. It requires the following:

- Alignment with mission and business goals.
- Instantiation of principles of SOA governance.
- Evaluation of relevant technologies

for SOA implementation.

- Recognition that SOA requires a different mindset than traditional development. ♦

References

1. Tilley, S., et al. "On the Business Value and Technical Challenges of Adopting Web Services." *Journal of Software Maintenance and Evolution* 16 (2004): 16, 31-50.
2. Schulte, R. "Meeting the Challenges of SOA Adoption." SOA in Action Virtual Conference, Nov. 2006.
3. "SOA Trend Survey." *InfoWorld* 2006.
4. Lewis, G., and L. Wrage. "A Process for Context-Based Technology Evaluation." Technical Note CMU/SEI-2005-TN-025. CMU/SEI, 2005.
5. Lewis, G., and L. Wrage. "Model Problems in Technologies for Interoperability: Web Services." Technical Note CMU/SEI-2006-TN-021. CMU/SEI, 2006.

About the Authors



Grace A. Lewis is a senior member of the technical staff at SEI. She is currently the lead for the System of Systems Engineering team within the Intermediate Systems to Intermediate Systems initiative. Lewis' current interests and projects are in SOA, legacy system modernization, and software development life-cycle activities in systems of systems. She has a bachelor's degree in systems engineering and an executive masters of business administration from Icesi University in Cali, Colombia, as well as a master's degree in software engineering from CMU.

SEI
4500 Fifth AVE
Pittsburgh, PA 15213
Phone: (412) 268-5851
Fax: (412) 268-5758
E-mail: glewis@sei.cmu.edu



Dennis B. Smith, Ph.D., is a senior member of the technical staff and is lead of the Integration of Software-Intensive Systems initiative at the SEI.

This initiative focuses on developing and applying methods, tools, and technologies that enhance the effectiveness of complex networked systems and systems of systems. Smith has been involved with working with DoD organizations in developing an SOA capability, including issues of SOA strategy, governance and migration of legacy assets to SOA. He was the co-editor of the Institute of Electrical and Electronics Engineers and International Organization for Standardization-recommended practice on Computer-Aided Software Engineering Adoption, and has been general chair of two international conferences. Smith holds a masters degree and doctorate from Princeton University, and a bachelor's degree from Columbia University.

SEI
4500 Fifth AVE
Pittsburgh, PA 15213
Phone: (412) 268-6850
Fax: (412) 268-5758
E-mail: dbs@sei.cmu.edu

CROSSTALK
 The Journal of Defense Software Engineering

Get Your Free Subscription

Fill out and send us this form.

517 SMXS/MXDEA

6022 FIR AVE

BLDG 1238

HILL AFB, UT 84056-5820

FAX: (801) 777-8069 DSN: 777-8069

PHONE: (801) 775-5555 DSN: 775-5555

Or request online at www.stsc.hill.af.mil

NAME: _____

RANK/GRADE: _____

POSITION/TITLE: _____

ORGANIZATION: _____

ADDRESS: _____

BASE/CITY: _____

STATE: _____ **ZIP:** _____

PHONE: (____) _____

FAX: (____) _____

E-MAIL: _____

CHECK BOX(ES) TO REQUEST BACK ISSUES:

- MAY2006** ☐ TRANSFORMING
JUNE2006 ☐ WHY PROJECTS FAIL
JULY2006 ☐ NET-CENTRICITY
AUG2006 ☐ ADA 2005
SEPT2006 ☐ SOFTWARE ASSURANCE
OCT2006 ☐ STAR WARS TO STAR TREK
NOV2006 ☐ MANAGEMENT BASICS
DEC2006 ☐ REQUIREMENTS ENG.
JAN2007 ☐ PUBLISHER'S CHOICE
FEB2007 ☐ CMMI
MAR2007 ☐ SOFTWARE SECURITY
APR2007 ☐ AGILE DEVELOPMENT
MAY2007 ☐ SOFTWARE ACQUISITION
JUNE2007 ☐ COTS INTEGRATION
JULY2007 ☐ NET-CENTRICITY
AUG2007 ☐ STORIES OF CHANGE
 TO REQUEST BACK ISSUES ON TOPICS NOT LISTED ABOVE, PLEASE CONTACT <STSC.CUSTOMERSERVICE@HILL.AF.MIL>.

Defining Services Using the Warfighter's Language

Michael S. Russell

General Dynamics Information Technology

Warfighters establish relationships with other warfighters to exchange information and accomplish their mission. Net-centricity and other information service concepts provide a means, but to a warfighter these are just technical solutions to the operator's need. The missing piece in the push towards service oriented approaches is an understanding of what warfighters expect from a service and a means to capture these expectations as part of the Joint Capabilities Integration and Development System (JCIDS).

Service-Oriented Architectures (SOA), and services in general, are a troublesome subject for many warfighters. Warfighters understand what information they need to execute a mission, and they are comfortable defining information requirements through JCIDS documents and supporting architectures. However, when this information is provided by a service, especially when depicted as a cloud labeled *Global Information Grid (GIG)*, this comfort level goes down.

Defining a service from a technical perspective is not hard. According to the World Wide Web Consortium [1], a service is a software system designed to support interoperable machine-to-machine interaction over a network. Services are standards-based and may contain the business logic needed to turn raw data into information.

This definition is fine for the engineers, but really does nothing to help warfighters specify what information they need from the service, when they need it, and how they can trust that the service – which they do not control – will provide them accurate and timely information. Additionally, the warfighter may not have visibility into, or control over, the business rules used to derive this information from raw data.

The issue with services is fundamentally one of perception. The average person typically goes to a Web site known to be trustworthy, one that provides timely and accurate information. Warfighters do the same thing; they get information from a trustworthy source, normally a higher echelon that they know provides

authoritative, timely, and accurate information. The services paradigm can break this approach by obscuring the source of information (and how that information was derived) from the warfighter.

When services are implemented in a way that mimics traditional lines of communication, nothing changes from the warfighter's perspective. They are still receiving the same information from the same sources. However, this approach

“An optimal services implementation allows the warfighter to specify the needed information, when it is needed, the quality of the information, and an objective way to determine the trustworthiness of the information.”

limits the benefits of a true services-oriented approach since it just replaces one communications technology for another without changing how information is discovered and delivered.

An optimal services implementation

allows the warfighter to specify the needed information, when it is needed, the quality of the information, and an objective way to determine the trustworthiness of the information. This allows the system supporting the warfighter to search the GIG and other networks for an available information source that provides that information, to establish a connection, and to start providing information. This optimal implementation will also enable the warfighter to specify, or at least have insight into, the business rules used to generate the information.

However, this implementation often requires the warfighter to accept that formalized information flow between command echelons, such as the flow of logistics information from superior to subordinate, may no longer exist. Telling a warfighter that his subordinates will receive information directly from a service outside of the chain of command may fit reality, but it may not fit doctrine or a warfighter's perception about how information should flow.

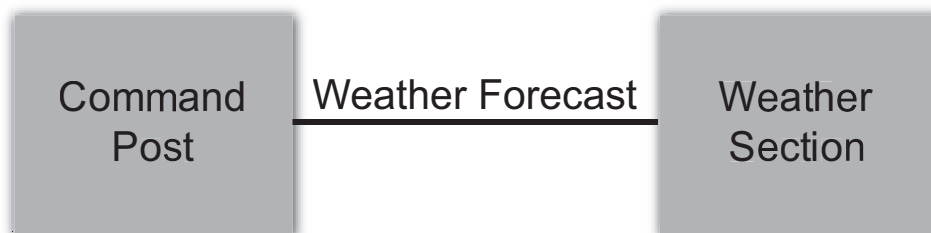
Defining Services for the Warfighter

Warfighters tend to think very directly: *I need to know tomorrow's weather forecast. The weather service provides forecasts, so I will call it and get tomorrow's forecast.* The warfighter could describe this need architecturally, as shown in Figure 1. In the example, the unit's commander knows that the same weather service is habitually attached to his unit and knows it will provide accurate and timely information. He will also have visibility into the business rules used to derive the information he needs from raw data.

Figure 2 shows a non-services implementation of this requirement. The network device could be a local area network, a radio, or another communications device.

A services approach could be used to implement the same technical requirement. In Figure 3, the technical solution

Figure 1: *How the Warfighter Describes a Need*



has implemented a services approach but from the warfighter's perspective, nothing has changed. The same information is flowing from the same source. This approach is similar to how selected services, such as address books, are currently provided.

Figure 4 depicts a common technical implementation that allocates the weather information requirement to a GIG service, as opposed to a habitually associated weather section. Several different weather information providers could conceivably meet the warfighter's information need. This depiction, even if technically correct, does not tell the warfighter who will be providing the information, only that someone will provide it.

Figure 4 highlights the issue many warfighters have with services. While the services themselves are invisible to the warfighter, the information provided by them is not. The issue from the warfighter's perspective is not about the services themselves, but rather the concept of who owns the information, how that information flows, and what rules were used to derive that information. This issue readily becomes apparent when a services approach is implemented in such a way as to break traditional command and unit relationships.

This approach has operational benefits: a deploying unit may no longer need an associated weather section, weather information can be pulled from a wider variety of weather sources, and constant weather information can be sent to all units within a geographic region. There are, however, downsides including quality of service, the loss of connection to a local information source, the need to quantify information trustworthiness, and the rules used to derive the data.

To mitigate these downsides, warfighters should have more control over the requirements process that specifies how services will be implemented. This includes being able to specify an organization that provides trusted information and the business rules that organizations use to generate the warfighter's mission critical data.

Requirements Development

Warfighters are responsible for defining information requirements by specifying who exchanges what information, with whom it is exchanged, why the information is necessary, how the information is derived, and how the information exchange must occur [2]. Typically, these information requirements revolve

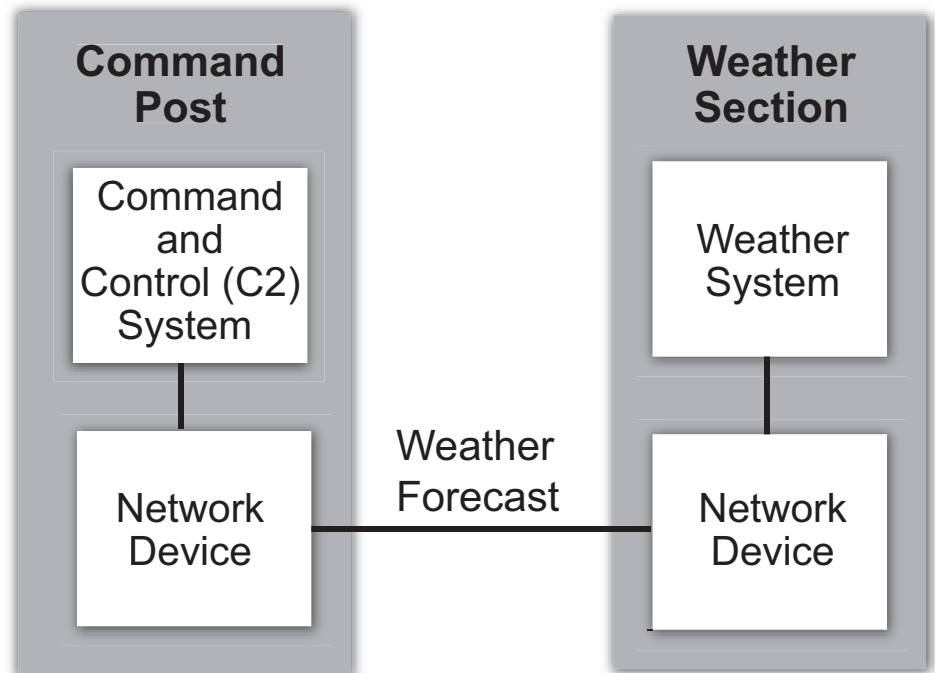


Figure 2: *Technical Implementation*

around the following:

- Describing the required information.
- Identifying the information producer and consumer.
- Describing operational performance, security, and information assurance attributes.
- Detailing the business process, including operational activities and/or triggers that initiate information transfer.

Describing how one operational unit sends information to another operational unit is straightforward. Using the weather forecast example, the weather section exchanges weather information with the command post upon request, the information is needed for mission planning, and the information is unclassified but must be securely and accurately delivered.

Continued on Page 18

Figure 3: *Service-Enabled Technical Implementation*

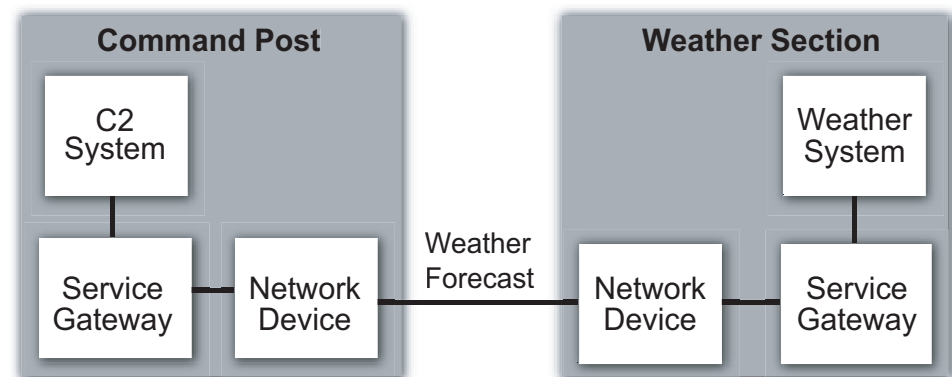
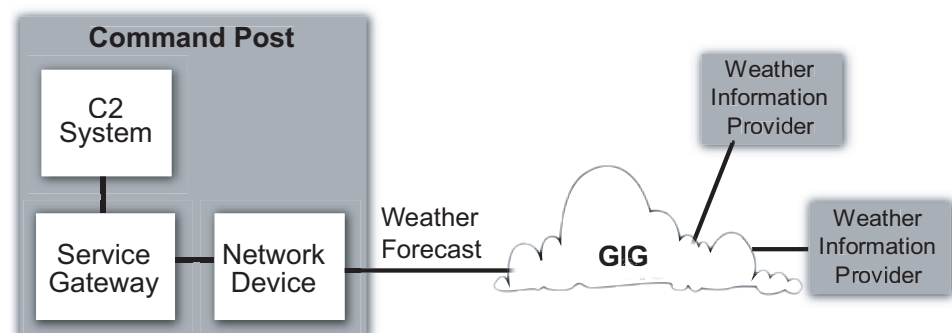


Figure 4: *Net-Centric Technical Implementation*



Systems and Software Technology Conference 2007: Enabling the Global Mission

The Systems and Software Technology Conference (SSTC) held June 18-20 in Tampa, Florida focused its attendee's attention on the theme of *Enabling the Global Mission*, striving to create and perfect the way military and industry professionals enable the warfighter by providing improved, accessible capabilities. Partnering with U.S. Central Command, the conference planners sought to emphasize and answer the needs of the warfighters by creating a collaborative and educational environment to explore proven best practices and share successful insights. In 2007, the SSTC departed from Salt Lake City after 18 successful years and landed on the Tampa Bay shores.

The central theme of *Enabling the Global Mission* streamed throughout the conference, facilitating open discussion among military and professionals to not only recognize the needs of the warfighter but to also address them. The SSTC conference continued its lifelong goal of creating an open environment to achieve results.

In his opening general session, keynote speaker Major General Timothy F. Ghormley, Chief of Staff, U.S. Central Command, presented attendees with a unique view of how software intensive systems are making a difference in our nation's defense, while integrating the needs of the warfighter through their perspective in the field. Ghormley challenged the audience to assist in developing ways to render improvised explosive devices ineffective and emphasized the urgency of this need. Ghormley repeated, "Any Network, Any Device" to enunciate the impending necessity of ease of communication.

Throughout the four-day event, government and industry leaders enlightened their audiences with solutions to important issues, such as focusing on system engineering problems early in product development and the need to find better ways to analyze computer code. Chris Miller presented an outstanding discussion of how we are living in exponential times. Jim Tucker expounded upon the horrific events of the attempted aircraft hijacking of Fed Ex flight 705, which he endured. The previous examples are a microcosm of the depth and spectrum of knowledge shared at SSTC 2007.

From the more than 120 presentations, attendees were able to delve into the topics of rapid response capabilities, robust

engineering, systems assurance, technology futures, communication infrastructure, and enablement of the workforce. Numerous insights were provided from projects and organizations, showing and educating on the success in their efforts. Common discussions within many of these presentations outlined varying perspectives on vulnerabilities and threats to software systems and countermeasures for them. Surprisingly we learned that real-time JAVA extensions are making the language an acceptable tool for real-time software development.

Attendees also discovered how they can use the newly released update to the Capability Maturity Model (Vers. 1.2) to improve their development processes.

In addition to presentations and tutorials, attendees also had the opportunity to participate in a panel discussion with government and industry leaders, providing questions and receiving answers that expressed the different perspectives from these two sides. This year's discussion sparked quite a lively debate between the panelists.

As has come to be expected with SSTC, participants were pampered with great food, key information, networking, and a fun evening cruising around Tampa Bay. Speaking of fun, attendees of this year's SSTC trade show were able to gain a sense of what it is like to fly though the Grand Canyon at near supersonic speeds in an actual flight simulator.

To sum up the SSTC experience, one appreciative and excited attendee noted, "I appreciated the real-world case studies provided by speakers and I received great information from a subject matter expert who has done it." Some of our other favorite quotes included, "Good social activities and opportunities to network," and "Great conference." Another attendee said, "I'm only allowed to attend one conference a year and SSTC is always the one I attend."

Although SSTC amended many traditions, it remains dedicated to preserving what makes SSTC the premier technology event for the Department of Defense by providing a showcase for the cutting-edge technologies. SSTC aims to create a collaborated environment to *Enabling the Global Mission* and serving the warfighter's need for ease of information.

Photo credits: Brent Baxter and Nicole Kentta

*"I appreciated the
real-world case studies
provided by speakers
and I received great
information from a
subject matter expert
who has done it."*

*"Good social activities
and opportunities
to network ..."*

*"I'm only allowed to
attend one conference a
year and SSTC is always
the one I attend."*



Nicole Kentta and Glen Luke greet trade-show attendees as they stop at the SSTC booth.



Barry Boehm, Jo Ann Lane, Mike Phillips, and Rick Turner discuss new article ideas at the annual CROSSTALK author luncheon.



Session attendees and speakers mingle before their morning presentations.



Conference attendees take a break to mingle and enjoy the fabulous food.



Jeff Schwalb poses for a picture between sessions at the trade show.



Attendees and guests board the yacht, Starship, for the conference's special dinner cruise event.

Continued From Page 15

However, when this same scenario is detailed using the services implementation in Figure 4, the information producer cannot always be defined. The information producer could be one of many weather information producers located in the deployed theater, at a stateside-based location, or any point in between. Instead of describing the expected point-to-point information exchange, the warfighter now sees a request sent to a service. This can leave the warfighter feeling he has lost control of where he receives his information and with doubts of the trustworthiness of the information.

Mitigating these concerns requires changes to the way operationally related architecture data is documented. First, change the way architectural diagrams are designed, highlighting the fact that an information service is provided by an organization, not a computer. Warfighters build trust with other warfighters, not the technology which provides information. Fundamentally, services are just the way an organization – no matter where in the world it is located – can provide information to the warfighter.

Second, requirements developers should add additional information to the warfighter's information exchange matrix (information concerning quality, trustworthiness, etc.) and to the warfighter's list of Information

Exchange Requirements. For example, trusted operational sources for information could be added as well as the warfighter's expectations for graceful information degradation on limited bandwidth networks.

Third, the warfighter is concerned about how raw data is captured, validated, and used to develop information. In today's rapid-paced environment, the warfighter does not have time to study raw data to derive the information needed to make effective decisions. However, making decisions based on information derived to support someone else's business process may lead to a bad decision. So the warfighter must be able to help specify the processes used to develop the information, or at least be able to have some visibility into the process.

Capturing the Warfighter's Service-Related Requirements

From the warfighter's perspective, information is being exchanged from one organization to another. From this perspective, services are the technical implementation – not the operational requirement. A diagram such as Figure 5, while it still includes the GIG, clearly identifies to the warfighter where he can get his weather forecast and, to some extent, an expectation about the quality of data that will be received; more importantly, it informs the warfighter who to call if the information does not

meet his mission needs.

With this diagram, the warfighter has a definite knowledge about who will be providing the information required to execute the mission (in this case a U.S. Navy [USN] stateside-based weather service and a U.S. Air Force [USAF] theater-based service). There is no need to capture every possible information provider, just the most likely ones. Figure 6 provides one possible technical implementation of the same operational need.

Approaching the architectural diagrams in this manner enables the warfighter to be more specific about what information is expected to be provided. This specificity should also be captured in the information exchange matrix. The Department of Defense (DoD) Architecture Framework v1.5 provides a recommended information exchange matrix format. The matrix emphasizes the operational characteristics of the information and is not intended to be an exhaustive listing of all operational details. Rather, this product is intended to capture the most important aspects of selected information exchanges [2].

Starting with this matrix and adding the following data elements will help capture the warfighter's expectations for service-provided information. These elements address the warfighter's need to control how service-derived information is sourced, stored, and acted upon.

- **Information owner.** The sending operational node is not necessarily the owner or producer of the information; sometimes it acts as a *pass-through* or *operate off* of derived information. This is the operational entity actually producing the information that the warfighter requires to execute his mission.
- **Information storage.** Does the warfighter require his system to locally store the information? Information such as operations orders may need to be stored, but individual common operational picture elements may not. Warfighters have mission critical information that must be present regardless of network or service operational status, and only being able to access this information when the *network is up* is not an option.
- **Information perishability.** How long does the information received from a service need to stay operationally relevant to meet the warfighter's mission requirements? If

Figure 5: Operationally Focused SOA Diagram

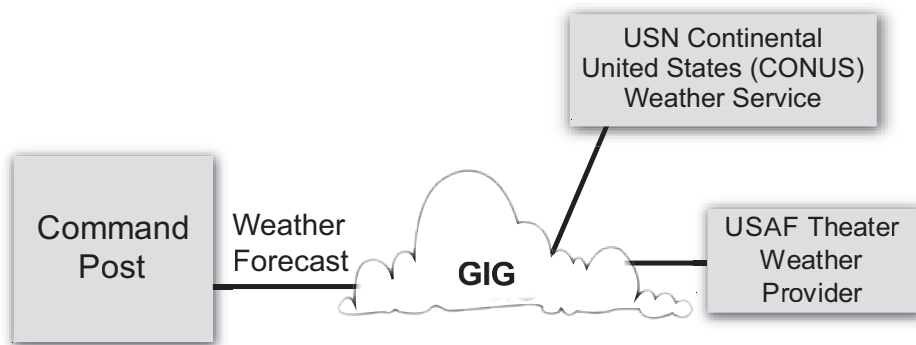
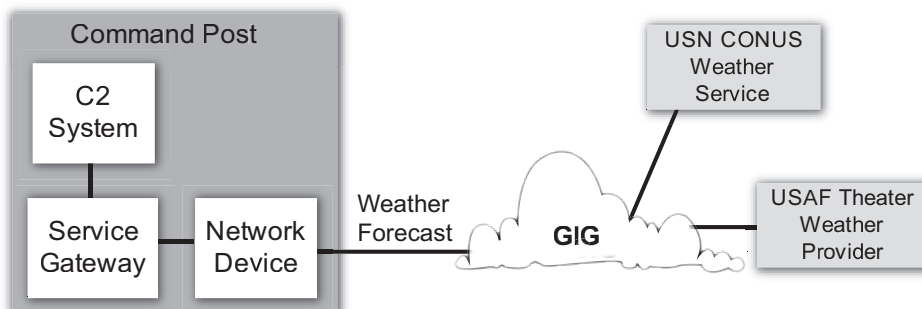


Figure 6: Implementation Oriented SOA Diagram



the network only allows a 30 minute information update, but the information is only valid for 15 minutes, the warfighter's needs will not be met.

- **Quality.** This is a subjective or objective measure that defines *how good* the information should be regardless of its source. Operationally, the information must meet this threshold; the system must be able to cycle among available, trusted information sources to provide the best information.
- **Trustworthiness:** This provides a subjective or objective measure that ranks the information owner, enabling the warfighter to give precedence to one information owner over another based on operational considerations. The warfighter should also be able to explicitly exclude information providers if desired.
- **Information derivation:** This includes the business rules the warfighter expects to be implemented to change raw data into useful information. There are many possible and correct ways to derive information. As the warfighter will be making decisions based on the information, not raw data, the warfighter needs to be assured that the way information will be generated meets its requirements.

The following data elements should be added to the matrix to capture how the warfighter's expectations about information produced by its system could be made available through a service. Though similar to some information assurance-related elements, they serve a different operational purpose.

- **Service discoverability.** Should the information be discoverable by all users (public), users fitting a certain profile (restricted), or only upon invitation (private)?
- **Subscriber roles.** Should information exchange be restricted to other users within a defined chain of command (role based) or not (non-role based)?
- **Subscriber availability.** Will the information be available to external subscribers continually (continuous) or in accordance with the situation and doctrine (limited)? This enables other warfighters to determine how much they can depend on this information source.
- **Subscriber storage.** Should the information be storable by the sub-

scriber? In some instances, there is an operational requirement to disable a subscriber's ability to store service-provided information.

Current Implementation

Weather information, a subset of what is known within the DoD as Meteorology and Oceanography (METOC) information was chosen to showcase a net-centric service currently available to warfighters called the Joint METOC Data Services Framework (JMDSF) [3] provided by the Naval Oceanographic Office or NAVOCENO. The JMDSF is designed to be a tool kit for deploying data-oriented services to securely deliver geospatial information and is the vehicle used by DoD to establish a single access point for all METOC data.

Using JMDSF, NAVOCENO can collate METOC data from numerous government and commercial providers, normalize this data, and publish it for all DoD METOC information users. JMDSF is responsible to the warfighter for recording where authoritative data resides, thereby easing the warfighter's concern for authenticating data [3]. To make this service even more useful, a Web-based front end and file transfer protocol push capability has been developed to enable the warfighter's system to retrieve METOC information in a variety of ways over networks of differing capacity.

No matter how useful, JMDSF is still a technical implementation of a net-centric service, not the operational activity providing the information. That organization will still be NAVOCENO, the Air Force Weather Agency, or some other operational entity. No matter how good JMDSF is at delivering METOC data on behalf of NAVOCENO, it is still only the current technical solution. The operational need to retrieve and act on METOC information does not change. By describing this need in terms of the operational requirement via the technical service solution, the warfighter can clearly define his requirements and be assured the technical solution will implement it.

Conclusion

Net-centric services provide warfighters with improved access to the information they need to make decisions, but only when these services are implemented in a way that reflects the warfighter's requirements. The services themselves are invisible to the warfighters, but the information the services provide is not.

Warfighters must be assured that their information needs will be met regardless of the technology that implements their requirements; especially since these services will be eclipsed by newer technologies over time. So identifying netcentric service requirements should be accomplished early in the JCIDS cycle and validated at each step. Regardless of the technology that provides information, the warfighter's requirement to ensure all information is timely, accurate, relevant, and trustworthy will not change. ♦

References

1. "Web Services Glossary." [W3C <www.w3.org/TR/ws-gloss/#web-service>](http://www.w3.org/TR/ws-gloss/#web-service).
2. DoD Architecture Framework Working Group. "DoD Architecture Framework, v1." Washington: Assistant Secretary of Defense for Networks and Information Integration, 2004.
3. Washburn, P., and T. Morris. "NAVOCENO Web Services: Online Data and Functionality for the Warfighter." *CHIPS* Jan.-Mar., 2005: 37-39.

About the Author



Michael S. Russell is a senior technical director for General Dynamics Information Technology. He has served as lead

systems engineer or systems architect on numerous federal, DoD, and industry development efforts, and currently manages programs for the U.S. Marine Corp's Systems Command. Prior to this, Russell served in the U.S. Army. He is a faculty member with the Federal Enterprise Architecture Certification Institute and is a member of International Council on Systems Engineering. Russell holds a master's degree in software engineering from George Mason University and has taught systems engineering courses for the past eight years.

General Dynamics Information Technology
16 Center ST
STE 109
Stafford, VA 22556
Phone: (540) 657-5393
E-mail: mike.russell@gdit.com

Applying a Service-Oriented Architecture to Operational Flight Program Development

Mitch Chan

309 Software Maintenance Group, Hill Air Force Base

This article describes how a Service-Oriented Architecture (SOA) was successfully applied to reuse data and applications previously deployed in single-user, single-computer configurations. The collection of data and applications was transformed into a unified, multiuser, client-server platform through the use of open source and standards-based technologies to minimize development time, maximize interoperability, and facilitate collaboration. The collaboration resulting from the multiuser network capability and shared resources enabled progress towards the Aerospace Basic Quality System Standard (AS9100) goals of lowering cost, meeting schedule, improving quality, and fostering a culture of continuous improvement.

This article describes the experiences of the Hill Air Force Base software engineers in the 309 Software Maintenance Group (SMXG) who develop the Operational Flight Program (OFP) for the Fire Control Computer (FCC) of the Block 30 F-16 aircraft. This article details how the engineers incorporated an SOA to automate its development and evaluation process of weapon coefficients for the delivery of air-to-ground munitions.

The use of open source and standards-based technologies were key success factors for a small team to accomplish the SOA implementation in a do-it-yourself fashion. Using publicly available open source software was instrumental in minimizing the implementation time and associated interruptions to the engineers' normal OFP candidate workflow. Choosing technology based on open standards ensured that they were maximizing the interoperability between systems.

The engineers were successful in reusing data and applications previously deployed in single-user, single-computer configuration and transforming them

into a unified multiuser client-server platform that resulted in a building-wide network capability. As a result, non-located system engineers, developers, and testers had access to the design and evaluation tools.

The improved collaboration resulting from the orchestration of network applications and shared resources enabled the engineers to achieve a return on investment (ROI) and progress in meeting its 309 Maintenance Wing's AS9100 objectives of lowering cost, meeting schedule, improving quality, and fostering a culture of continuous improvement.

This article proceeds by first providing a background section that introduces the key terminology. Next, the article describes the former development and evaluation process. Then, the article highlights former inefficiencies and provides the solution which incorporated an SOA as an integration tool to achieve better results.

Background

1. What is a weapon coefficient?

Weapon coefficients are parameters used to calculate weapon trajectories.

Examples of weapon coefficients include weight, the cross-sectional area, and other aerodynamic performance factors.

2. What is a footprint?

For every weapon release there is an associated area on the ground representing the possible impact points. This area is referred to as the footprint.

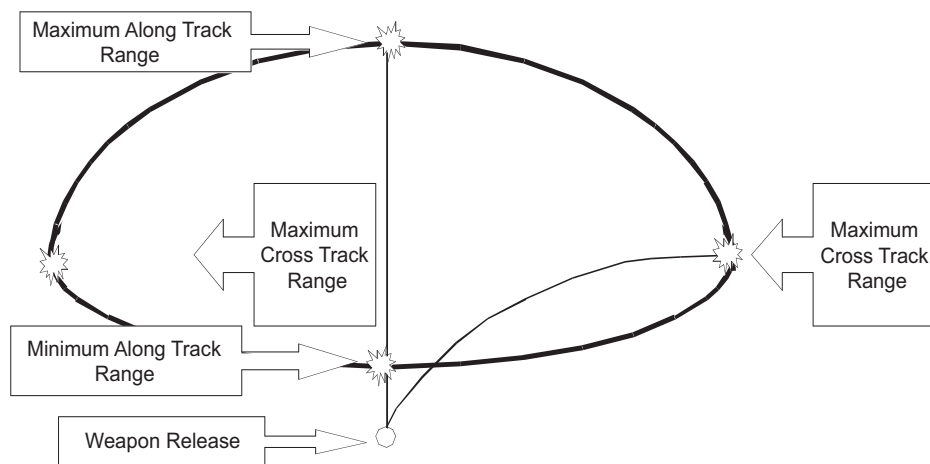
3. What is a flyout?

In particular, there are four points on the boundary of the footprint that are of interest: the Maximum Along Track Range, the Minimum Along Track Range, The Maximum Right Cross Track Range, and the Maximum Left Cross Track Range. These points correspond to the longest forward distance, the shortest forward distance, the furthest right-forward distance, and the furthest left-forward distance, respectively, that the weapon can achieve. These points are referred to as flyouts.

4. What is truth data?

Truth data is the real-world results used to compare calculated trajectories and impact points. Truth data is typically provided by the weapon manufacturer in the form of files or a computer application (weapon model) which produces an output file.

Figure 1: JDAM Flyouts



Former Weapon Coefficient Design Process

Figure 1 shows a top view of a Joint Direct Attack Munition (JDAM) release. In the FCC, the four JDAM flyouts are a function of the weapon release point and the weapon coefficients. The four flyouts define a quad-ellipse footprint.

Design Objective: Generate the optimal set of weapon coefficients to determine JDAM flyouts for all release points.

Figure 2 shows the former 3-stage weapon coefficient design process. In

stage 1, the system designer ran the weapon models to generate data for the purpose of comparing the results of the FCC calculations. In stage 2, the system designer generated a set of weapon coefficients intended to meet the design objectives stated above. In order to complete this task conveniently, the system designer used a simulation of the FCC OFP. Finally, in stage 3, the system designer handed off the weapon coefficients to the developer who incorporated the weapon coefficients into the OFP code. Both the developer and tester compared the test stand results against the simulator results used in stage 2.

Inefficiencies of the Former Weapon Coefficient Design Process

From a resource point-of-view, the former design process relied predominantly on the system designer, who performed stage 1 and stage 2. Because this process was *serial*, the developer and the tester were kept waiting until the end of design process. The system designer became the specialist and limited his *bandwidth* (available time and energy) to work on other projects.

The design process did not scale well. For practical purposes, the system designer addressed one weapon at a time, which caused a bottleneck if two or more weapons were involved.

Finally, and most importantly, the design process was not collaborative. The system designer essentially disappeared and the design of the weapon coefficients became a black art. The single-user, single-computer deployment did not encourage developers to invest time in the development tools.

Enter SOA

SOA Purpose

The SOA was designed, implemented, and launched with the intent of making the design tools and data accessible on UNIX desktops across the building network. Servers were a mixture of personal computer and UNIX workstations.

SOA Design

Figures 3 and 4 are the UML diagrams that describe the design of the SOA. Figure 3 shows the five scenarios or use cases. Figure 4 (see page 22) describes the orchestration sequence of the user's client application. Note that the fifth Use Case *Store Results* was added with the intent to replace data that was previously stored in personal directories with a cen-

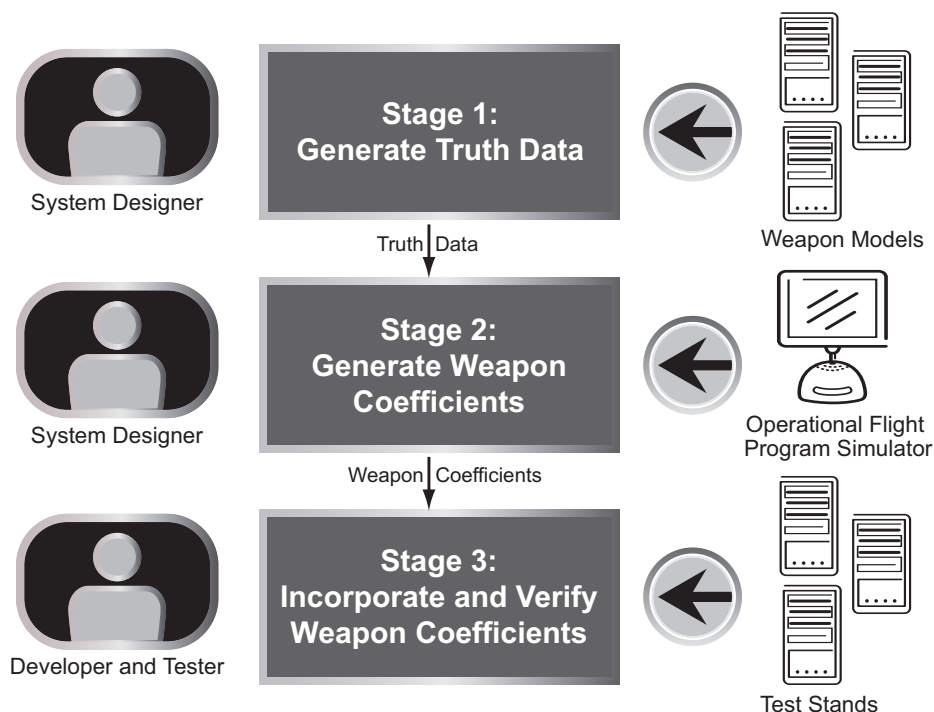


Figure 2: Former Design Process

tral storage repository.

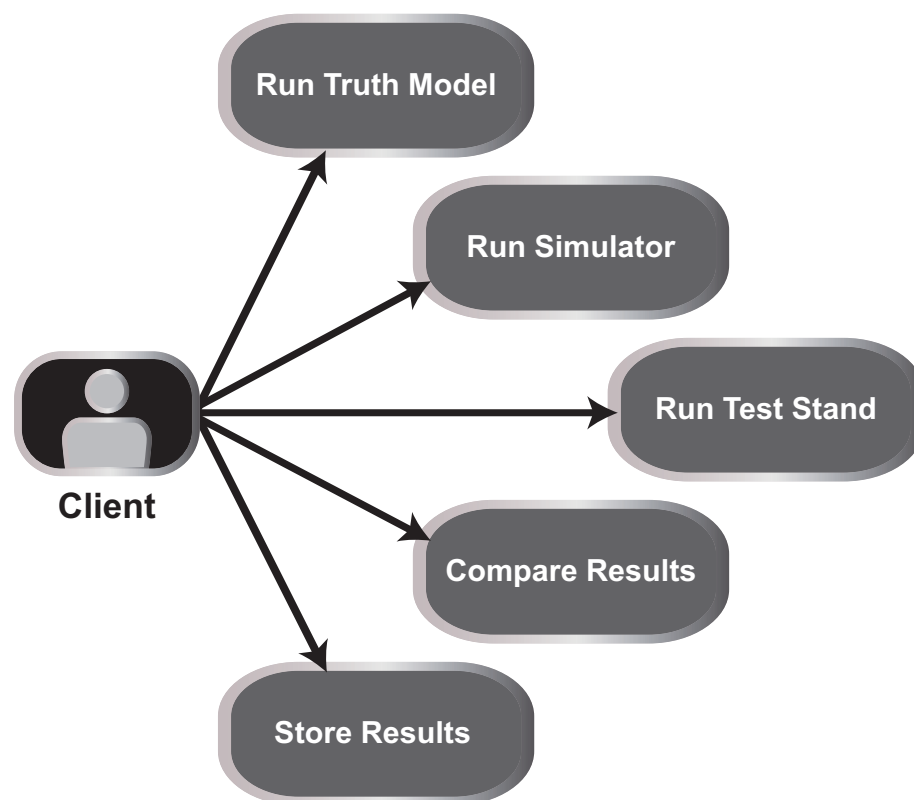
Figure 4 elaborates some details of the use cases discussed earlier. Note that results are stored in a database versus files.

SOA Implementation

Remote access to the database and the weapon models were provided using

open source Web Services. The Web services were constructed using two Java 2 Enterprise Edition (J2EE) Web servers. The weapon models were accessed using a SUN Microsystems Java Web Services Development Pack server. Data Access Objects were exposed across the network using an Apache Tomcat Web server with Axis to provide the Web services connec-

Figure 3: Use Case



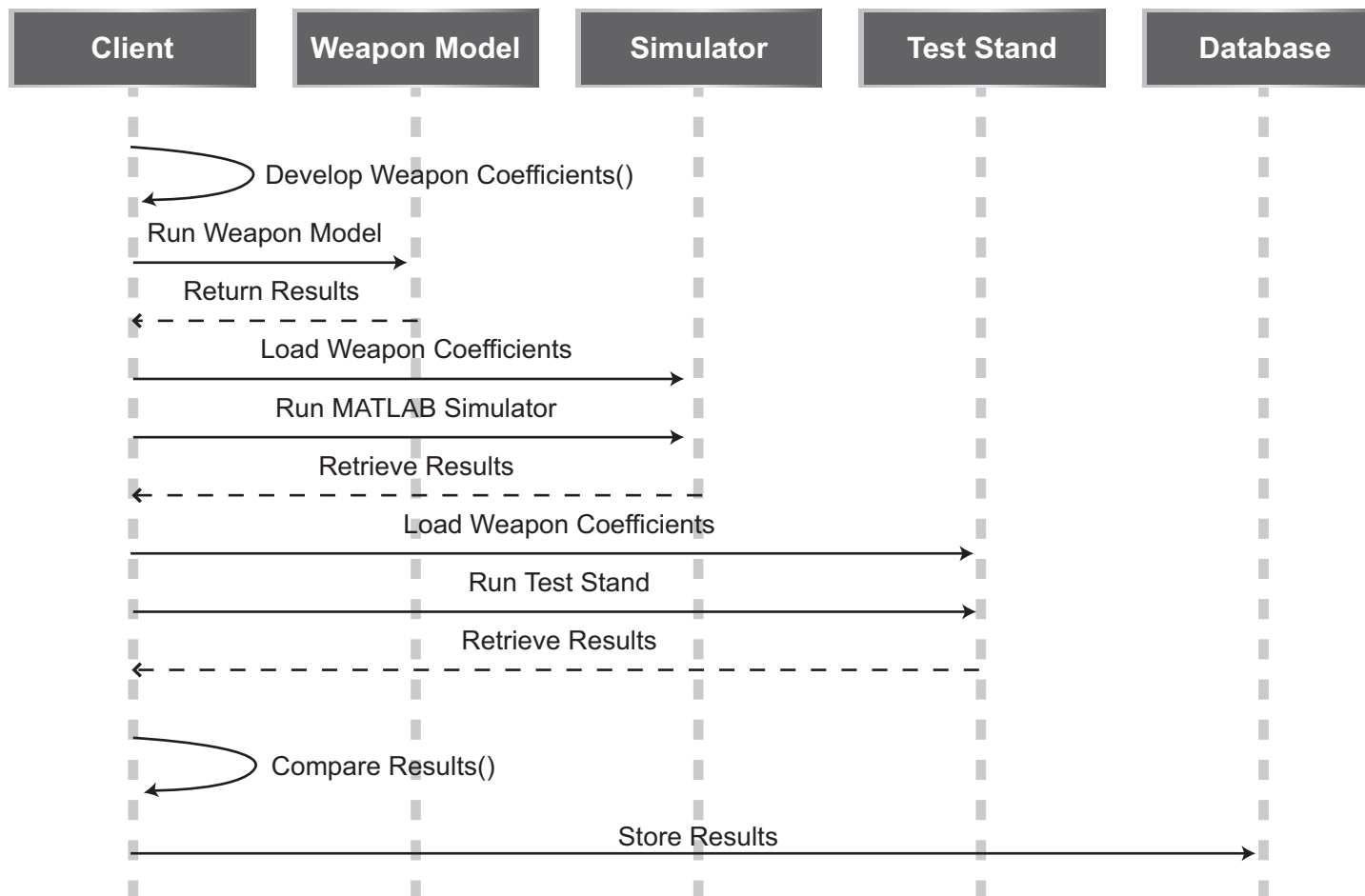


Figure 4: Sequence Diagram

tion, and Spring/Hibernate to provide the data access objects, persistence, object-relational mapping, and database connection.

The choice to use open source software greatly accelerated the implementation since a major coding effort was avoided. The majority of the effort was

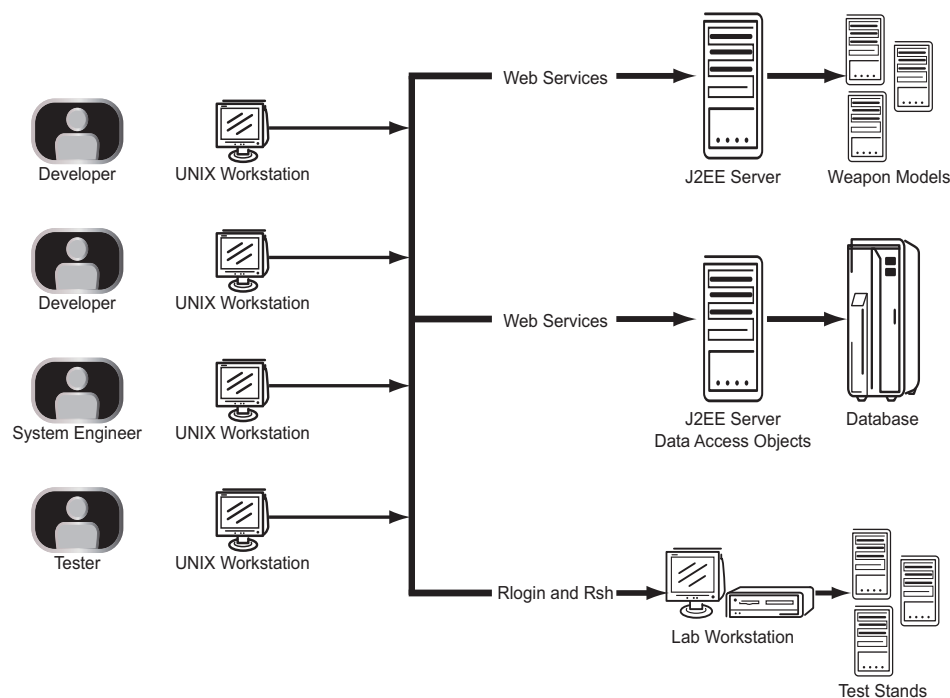
tweaking pre-existing Java source code and editing of eXtensible Markup Language configuration files. The use of standards-based technology on the server-side such as Web services ensured maximum interoperability and tools to create client applications¹.

The lab workstation running the test stands was accessed via UNIX Remote Login and Remote Shell Programming since the client and server workstations both ran UNIX.

Orchestration of the weapon models, the database, and the test stands was performed using Matrix Laboratory (MATLAB) in concert with Perl to create a rich client interface. MATLAB and Perl were chosen as client orchestration tools since they were native to the client Sun workstations.

MATLAB was chosen primarily because of its visualization tools, graphical user interface interfacing tools, math library, and toolboxes. Most of the engineers, especially the recent hires, had plenty of hands on experience with MATLAB. Perl was used for its regular expression capability to handle text input and output. Perl also had the ability with its Simple Object Access Protocol-Lite

Figure 5: Enterprise Diagram



(SOAP-Lite) package to create a Web services client simply by setting its service pointer to the URL of the Web Services Definition Language file.

Figure 5 shows the final Enterprise Diagram for the SOA. Note that the simulator does not appear since it runs locally on the client UNIX workstation.

SOA Innovations

20 Minutes to 20 Seconds

One of the first initiatives was to reduce the role of the simulator and develop directly on the test stand. The major hurdle was the length of time required to check whether the coefficient changes were right or wrong. Coefficient changes had to be incorporated in the OFP code. Then the OFP code had to be re-compiled which took more than 20 minutes.

The contractor responsible for the test stands recommended and implemented a workaround to speed up the process. The contractor added a capability to upload weapon coefficient files which took only 20 seconds. The new process of incorporating coefficient changes in a file and uploading the file was simple and fast. With the new process, it became practical to develop directly on the test stand.

Multiple Sessions on the Desktop

Using an older version of MATLAB (Vers. 5 versus Vers. 14) enabled multiple MATLAB client sessions to run on the UNIX desktops. This enabled multiple projects to be open at one time. Perl, in concert with its SOAP-Lite package, facilitated the use of an older version of MATLAB (Vers. 5) which did not have the Web services remote access capability but was less of a central processing unit and memory hog.

ROI and Progressing Towards the AS9100 Objectives

The AS9100 organizational objectives are the following:

1. Decrease cost.
2. Meet or exceed schedule.
3. Improve quality.
4. Develop a culture of continual improvement.

Decreasing Cost

Cost saving was realized both short term and long term. In the short term, the SOA was applied to the current release of the FCC OFP. Expensive end-of-cycle rework costs were avoided by getting the design right the first time. In the long

term, cost saving was realized by reduced man-hours (50 percent reduction) resulting from the improved automation and efficiency. A fellow engineer working on a follow-on project noted the following: *I love the tools. I can run the application, check back later, and find all the graphs and results that I need.*

Meeting and Exceeding Schedule Requirements

Removing the keystrokes and mouse clicks reduced the probability of operator error. Comparison between different weapon coefficients sets and improved collaboration lowered the *variability* between developers in designing weapon coefficients. These factors helped to achieve better *predictability* in the execution of the design process.

Improving Quality

The quality of weapon coefficients could be scored by *coverage* and *false positives*. *Coverage* is the percentage of the area of truth model footprint covered. *False positives* are the number of impact points lying outside the truth model footprint. Also, the number of release points on the test stand was increased by more than 1,000 percent. The previous method was too time consuming to allow scoring of more than 40 release points per set of weapon coefficients (over 400 coefficients tested at a time). The efficiency of the new method now allows us to score up to 1,100 release points per set of coefficients.

Developing a Culture of Continual Improvement

The SOA enabled a more data-driven design process. As alluded to earlier, the scores *coverage* and *false positives* could be measured from the test run data recorded in the database. Adding more release points over a full range of release conditions increased the statistical significance of these scores. Using the current scores as a feedback mechanism, the developer could further refine the weapon coefficients to produce better scores for the next design iteration.

Lessons Learned

The engineers considered a base-wide network version of the SOA. However, servers and applications on the base-wide network were subject to quarterly time compliance network orders and vulnerable to any collateral damage resulting from patch pushes to update the Microsoft Operating System and

Standard Desktop Configuration. In the end, the engineers decided to stay off the base-wide network to avoid the extra computer administration and maintenance.

Conclusion

This article presented SOA from the front line and trenches view of software OFP development at Hill Air Force Base. An SOA applied to automate software development process was introduced. An overview of the inception, elaboration, construction, and transition was covered. Finally, no discussion of an SOA would be complete without evaluating the ROI. ROI was presented in the framework of meeting the 309 Maintenance Wing AS9100 core objectives. ♦

Note

1. Deliverables from the Basic Profile Working Group. 2007. WS-I Web Services Interoperability Organization. 6 June 2007 <www.ws-i.org/deliverables/workinggroup.aspx?aspx?wg=basicprofile>.

About the Author



Mitch Chan has served four years at Hill Air Force Base as a Principle Engineer and an Embedded Systems Software Developer for the 309th

developing the OFP for the FCC of the Block 30 F-16 aircraft. He specializes in the delivery of air-to-ground weapons and was the primary FCC engineer responsible for adding the capability to deliver the 500-pound JDAM (GBU-38), currently employed by the F16C+ aircraft deployed in Iraq. Chan has a bachelor's degree in mathematics from the University of California at Berkeley, a bachelor's degree in electrical engineering from California State University at Sacramento, a master's degree in electrical engineering from Santa Clara University, and a masters of business administration from the University of California at Davis.

**309 SMXG
6137 Wardleigh RD
BLDG 1515 RM 248
Hill AFB, UT 84056
Phone: (801) 586-6756
E-mail: mitch.chan@hill.af.mil**



For Net-Centric Operations, the Future Is Federated

John Michelsen
iTKO, Inc.

SOA (Service-Oriented Architecture) is an ideal strategy for enabling net-centricity across shared application environments. But before materiel providers can realize the flexibility and reuse advantages of migrating systems to a shared SOA model, how can they avoid the risk of missed end-customer requirements? Achieving a sense of trust in SOA requires more than the right development and testing technologies. Net-centricity requires constant validation and a shared certification process to ensure that applications are meeting the needs of the warfighter – even as the shared technology environment changes and evolves.

Leading institutions are moving toward net-centric systems: highly distributed and flexible methods for delivering required technology in a shared, reusable way to support operations. Achieving net-centricity in the military and other government agencies will require both organizational policies for sharing a common set of goals and technology strategies like SOA.

However, there are good reasons why divisions often build and maintain their own monolithic applications: a lack of trust. How can you be sure that a third party will deliver the required functionality if they are not *on the line* to make it work for you? Realizing an effective federated SOA strategy in mission-critical theatres, such as warfighter scenarios or air traffic control, requires establishing *trust* between peer units with services managed outside of the vertical chain of command.

Technology innovation around SOA is happening on a rapid scale within both defense and civilian government agencies. This change is being driven by economic and operational concerns that the business world at large may not fully comprehend. Indeed, a typical business technologist may comment on the operational and budgetary *bloat* of federal organizations, but reality demonstrates that there is fierce competition for investment dollars within the public sector. Programs must reach milestones of success, despite resource and time constraints, and within a potentially even more dynamic environment than the typical commercial enterprise.

By pushing for long-term goals, the public sector provides value to taxpayers by helping create a *purpose-driven market* for technology investments. After all, where did we get the Internet itself (Defense Advanced Research Projects Agency-net [DARPA-net])? Or the quality

and implementation best practices of Capability Maturity Model Integration (CMMI) or the Information Technology Infrastructure Library (ITIL)?

The common goals of many government and defense agencies will create a purpose-driven market: a notional architecture where services can be published

“By pushing for long-term goals, the public sector provides value to taxpayers by helping create a purpose-driven market for technology investments – opportunities for innovation that the stockholder-driven company can never realize.”

and consumed through a central, shared authority and knowledge base while retaining the flexibility to meet operational and business needs.

Pertaining to the government as a model SOA example, Dave Linthicum of the Linthicum Group said the following:

I think that the government can benefit most from SOA, considering the nature of their business and the underlying need to have many systems interoperate.

However, there needs to be a realistic understanding of the issues at hand, and perhaps they need some new approaches other than building architectures that look like archeological layers of past IT contracts. I do think some of the more spectacular SOA successes will come from the government side. [1]

We can think of this system just like we think of the idea of multiple state governments operating within a federal government. Each state has its own laws, objectives, and budgets, but all of the states in the system are also governed through a federal authority. This model can also be applied to SOA application architectures, to define a *Federated SOA*.

The Federated SOA will provide a leading indicator for where net-centric software innovation is headed for the business world. The innovations that led up to DARPA or CMMI were not designed simply to make money or optimize cost. They were born from the idea that we have a specific, targeted outcome to reach. Let us take a step forward to prove why governmental approaches are going to produce leading-edge techniques for SOA governance.

Challenges

Federal agencies, just like any other global enterprise, are now at a crossroads for establishing an SOA strategy in a world where no single strategy can possibly cover every need. Countless siloed technologies currently exist as acronyms within each operational unit. Supporting and maintaining so many separate platforms becomes untenable over time, as any additional functionality or code adds to the existing technologies in a stovepipe fashion. Each new customization and every line of code written for one of these *stovepipe* technologies results in a long-term annuity that will have to be

* CMMI is registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

paid back and supported over time.

Organizational: Vertical Trust vs. Horizontal Competition

For net-centric computing environments, *vertical trust* (i.e. trust up and down a common chain of command) is far easier to achieve than *horizontal trust* (i.e. across authority domains or organizational units).

Vertical Trust

In most organizations, vertical trust is relatively easy to come by and already exists. The line of reporting structure typically creates the sense of reuse standardization and the ability to leverage vertically within an organization.

As shown in Figure 1, in vertical hierarchies within organizations, there is an expected level of shared trust.

- The higher levels within can expect the underlying teams to *build to order* technology assets and maintain them according to defined policies.
- Supporting providers can expect the requestor to leverage the developed materiel capability services according to well-understood and defined requirements.

Horizontal Trust

At the highest levels, horizontally across peer groups, a lack of trust is usually not the case (see Figure 2). Ironically, a lot of organizations that share common goals tend to foster a *not invented here* mentality that inhibits collaboration between horizontal peer groups. That is the source of why horizontal governance is such a critical aspect to SOA adoption.

Across different operational or business units, coordinating the proper use of a service can be difficult.

- Materiel or service providers want to establish reuse of their services, but they are answerable to different stakeholders.
- Upstream consumers of services may not provide clear enough use cases or policies of how they will leverage the services.
- Therefore, teams often build and maintain redundant functionality in vertical silos.

Redundant efforts create inefficiencies when software functionality gets duplicated. Therefore, before we can realize the value of reusable services offered within a federated software strategy, agencies must learn to establish trust horizontally, across organizational

Vertical Governance = Piece of Cake

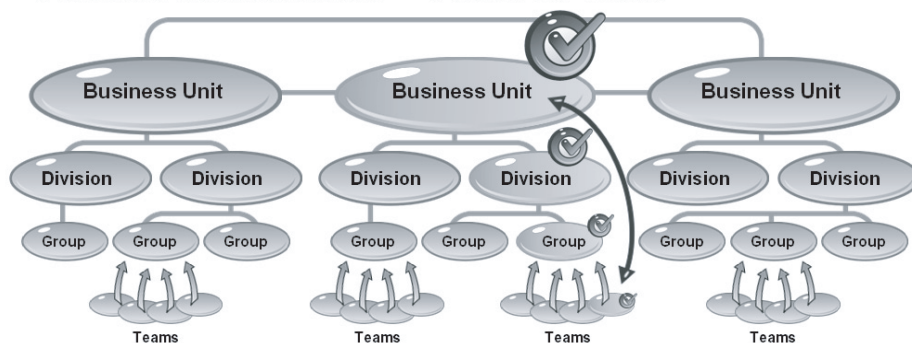


Figure 1: *Vertical Governance Along a Chain of Command*

and chain-of-command boundaries.

Federated Technologies Are Heterogeneous

The operational rules and behaviors of an SOA live in the middle tier (between the interface and the database layers), iterating within an *alphabet soup* of technology acronyms (for example, XML, SOAP, WSDL, ESBs, etc.), which are exposed as services. These services are technology assets that can be managed or published within your own authority domain or reside outside the department or even outside the organization.

Services Need Not Be Web Services

Many technology vendors simply equate Web services (WSDL/SOAP²) with SOA; from a testing point of view, they equate the testing of Web services with the testing of SOA.

While it is true that a number of initiatives for doing SOA are very Web services-centric, the Aberdeen Group's last research on this points out that only about 50 percent of the SOA initiatives at best-in-class companies are Web services-based [2]. There are a variety of technologies being used to create that commoditized middleware for SOA. While Web services can be a good integration

strategy, other technologies are valid and possibly better for a given organization than a Web services stack, for instance, using an Enterprise Service Bus with little reliance on Web services.

The distinction between Web services and SOA testing in general is important. There is more than one way to build an SOA. Teams need to test the implementation and side-effects that occur across heterogeneous technologies, as opposed to just a selected middleware layer like SOAP.

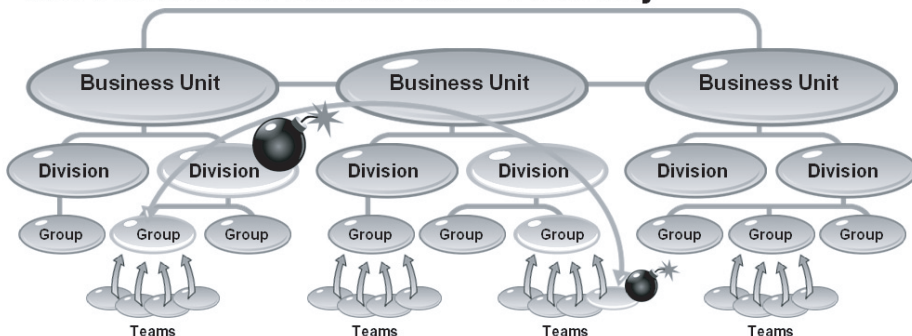
As illustrated in Figure 3 (see page 26), there is inevitable complexity under the surface of any large-scale implementation. In an SOA, *more unique technology types*, multiplied by *more points of connection*, equals an *exponential increase in possible failure points*.

Heterogeneous technologies will never go away and leave behind a totally homogenous platform. There are several reasons for this.

1. **Legacy systems cannot just be turned off.** Technologists are always attempting to provide more flexible application architecture at a lower cost to their constituents. As technologies evolve, we almost never have a cost justification to retire existing systems and replace them with the new technology. The Web services

Figure 2: *Horizontal Trust Across Organizational Boundaries Bears More Risk of Misunderstandings or Missed Requirements*

Horizontal Service Reuse = Anarchy



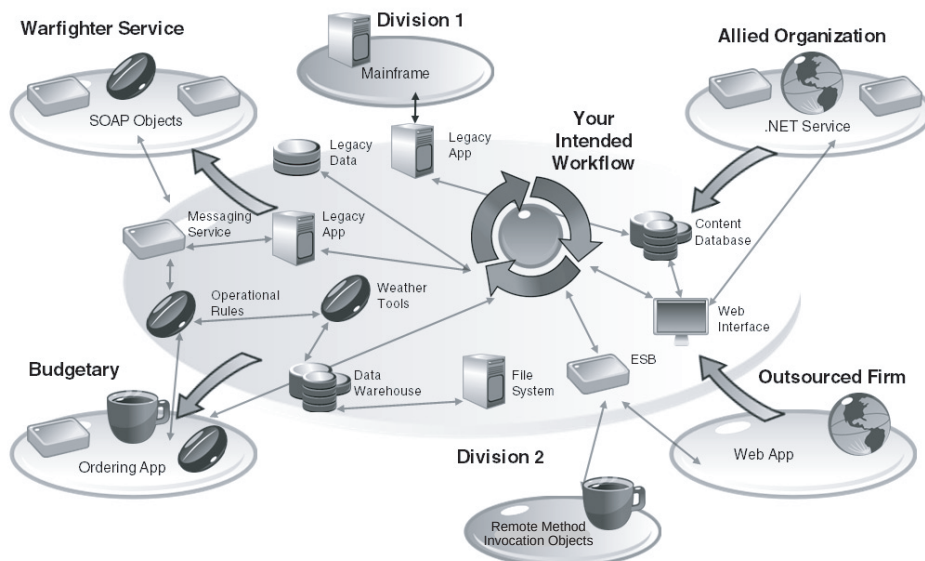


Figure 3: *Underneath the Expected Simplicity of a Created Net-centric Service-Based Workflow, a Myriad of Distributed Existing and Disparate Technologies Must Be Managed and Tested*

stack now considered modern technology will soon be outdated, and an integration strategy to leverage these soon-outdated applications with tomorrow's better thinking will be needed.

2. Resistance to vendor monopolization. A *we do it all* vendor can promise uniform specifications and the benefits of increased scale, but for larger clients, tying the entire technology to the platform of a single vendor may inhibit specialization and become perceived as a long-term risk if development priorities (or pricing structures) change. Thus, even new applications are built on a variety of technology platforms.

3. Distributed authority domains. Federated organizations have multiple chains of command. Operational units have unique functional needs from technology assets, therefore, they naturally desire to keep some teams and service assets under direct control.

To trust SOA, a much deeper level of collaboration testing must occur as a continuous process, not an event. The services, and the expected uses of them, must be submitted and certifiable (functionally and at load) to the community relying on the SOA.

Policies Are Hard to Follow

A policy basically defines an expected behavior. Governance of SOA applications goes hand-in-hand with defining and enforcing policies in order to achieve control of, and trust in, the SOA application.

But what is policy? Often technolo-

gists mistakenly approach policy as a solely structural concern. For instance, *is the syntax of the XML message formed correctly? Do the components connect according to our selected standards?* Structural concerns are just one aspect of policy that needs to be addressed.

Once structural standards are in place, a behavioral definition of policy will become a determining factor in the success of any SOA endeavor. A behavioral policy basically defines what the system should operationally do to support its intended use.

We need to realize that the organization will not want to write policy concerned about technical standards. Rather, the organization's version of policy will be the following:

- I really need functional integrity on this particular transaction activity of the system.
- I really need the response on resource availability to be accurate within 30 minutes of my request.
- I cannot allow stale data to be reported as current activity in the field.

Net-centricity is about sharing operational functions and placing expectations upon the systems that are implementing those policies. That is meaningful policy. SOA policy must focus on the functional integrity of the application – the quality and reliability of the end customer experience, accuracy of data, and runtime performance of the application.

Service Consumption Is Not Free

There is a commonly held (and somewhat sentimental) notion that once the SOA architecture is in place, it will pro-

vide an environment of published services that multiple consumers can basically leverage within their own workflows at little or no cost. This model is valid for very non-differentiated or commoditized services such as news feeds, simple *calculators* for unit conversions, and the like.

Take for instance a small application, developed specifically for a military unit's internal use. The certification level, and the level of structural, behavioral, and performance policy will not be nearly as high. The maintenance cost for that service will be much lower and there will be much less risk and testing rigor in changing that service when a finite set of consumers are impacted by that change.

Now consider a mission-critical, broadly reused service that will have a widely distributed use among all military divisions, including some that the development team may not even yet be aware of. That creates an increased cost of producing a service that is robust over time and reusable. If consumption creates cost and effort for the producer, then the consumption itself should not be free.

A reckoning must occur when a consuming project team gets benefit from an existing reusable service that is properly maintained. Otherwise, the production of robust, quality services is penalized. An increased cost burden, without an associated increase in the budget for that service to be reused, will actually threaten that service's long-term quality and adherence to the policy the consumer expected.

Consumption of existing services can reduce the cost and effort of producing a solution. Over time, the service consumer must bear some accountability to the publisher of those services. If all parties involved in SOA realize that there is no *free beer*, the end result will be a more sustainable marketplace of services.

Solutions

Establishing SOA Governance Enhances Capabilities

The primary goal of net-centric computing is to enhance the ability of each organization to efficiently meet the needs of the warfighter. If the level of trust is high, the organization can rely on both the historical and runtime validation of every service it depends on. Without SOA governance, SOA remains a chaotic, free-form exercise.

If the extended organization plans to overcome the *vertical silos* and rely on an application made from separately managed services, each being developed and maintained on their own life cycle, what are the rules of the road?

According to Gartner analyst Frank Kenney [3], SOA governance is made up of three components: the *Registry (or Repository)* where the assets of SOA are stored and catalogued, *Policy* which is meant to keep track of the *rules of engagement* and service levels expected in SOA, and *SOA Testing* that is needed to ensure SOA life-cycle quality.

What good is a registry if it contains assets that are not sufficiently tested at both the service and the implementation layer? Strong testing is required to ensure that the SOA application continually meets the business needs – in development, integration, and deployment. The longer testing is delayed as an aspect of SOA governance, the wider the deviation becomes between expected and delivered results.

Defining a Big Policy

As we continue to mature the SOA governance space, the policy area appears to be the one that is the most immature. In the near future, governance will become synonymous with policy. Each type of SOA policy is vitally important to achieving reliability and trust, as shown in Table 1.

Currently, most technologists focus on testing the structural policy type mentioned in Table 1. True, integration standards are important, but once those types of problems are solved, behavioral and performance level validation will gain prominence. After all, what good is certifying structural policy if the SOA application does not perform its function correctly and at the expected scale, design time, run time, and change time?

The Certification Environment: Two Sides of the Coin

There are two critical certification flows to a robust, federated test and policy validation strategy: a publish cycle and a consume cycle. These can be considered federal-level processes that form a certification environment, a central authority offering the *rules of the road* for creating and offering services, then leveraging these created services in an expected fashion. The need for a publish cycle is evident: A standards body must establish and enforce criteria that provide for an environment of trust to encourage and enable reuse. The consume cycle is equally as important, as it must properly lay out the expected use cases for published services in a realistic and enforceable way.

The Publish Cycle

A development team that wants to offer up a service to the community submits

Policy Type	Definition	Examples of Use
Structural	The services components are compliant with chosen integration standards and reusable with the current development, deployment and governance platforms.	- Do the <i>pin outs</i> line up so that the components can technically communicate with each other? - Are the services following correct authentication protocols? - Is the XML syntax compliant?
Behavioral	The service interacts and provides correct results within the context of the workflow or task that needs to be accomplished.	- Are the results I expected actually being produced? - Does the operational logic of this service properly support the process it is being used for?
Performance	The service can sustain the performance, scalability, and reliability levels required over time.	- Can this component produce the results I need with the number of users I need, within the time constraints, and infrastructure that I need it?
Runtime	Expectations around the service level of the component in the live production environment.	- Expected response time for this service is one second for our top 20 constituents and three seconds for all others. - Uptime must be > 99.999 percent.

Table 1: *Defining Policy Types*

their asset as a proposed service for reuse (see Figure 4). This service is reviewed by a cross-domain group called a certification group that must verify that the offered service conforms to expected policies before making it available to the community. Next, the group needs to continuously monitor and test those services, as they may change or fall out of expected policy guidelines.

How the Publish Cycle Works

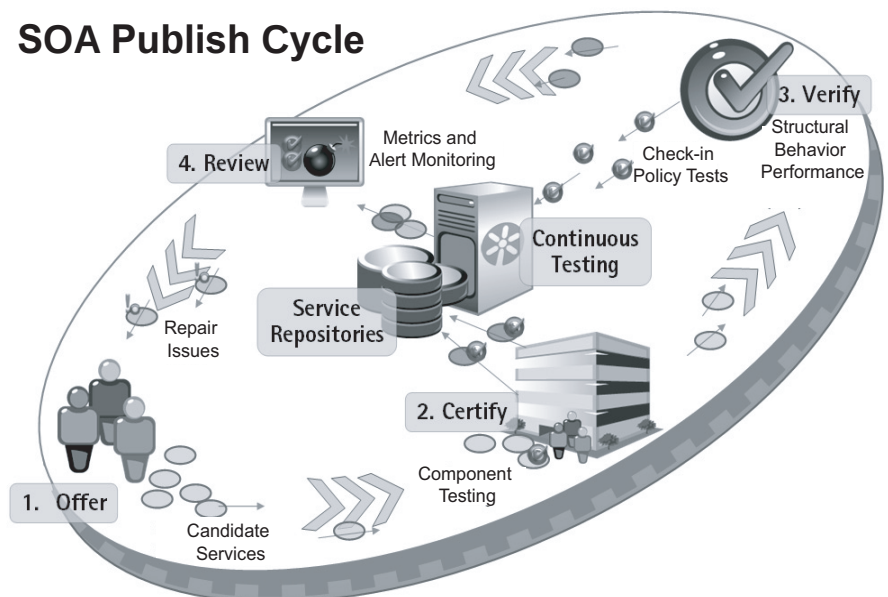
- 1. Offer.** A developer or development team creates a new, uncertified version of a Service Component (SC).

They can offer the working SC along with documentation to a certification group for testing and approval. Optionally, they can include working test cases as part of the documentation process to aid certification and store those in the registry.

- a. Developers self-assess quality by creating tests against their own SCs (whether it is a technology component or Web service) before submitting them.
- b. Developer offers the proposed SC to a centralized registry for certification. The development team can

Figure 4: *Publish Cycle for Service Providers to Submit and Certify That Their Service Assets Will Meet the Needs of the Net-Centric Community*

SOA Publish Cycle



SOA Consume Cycle

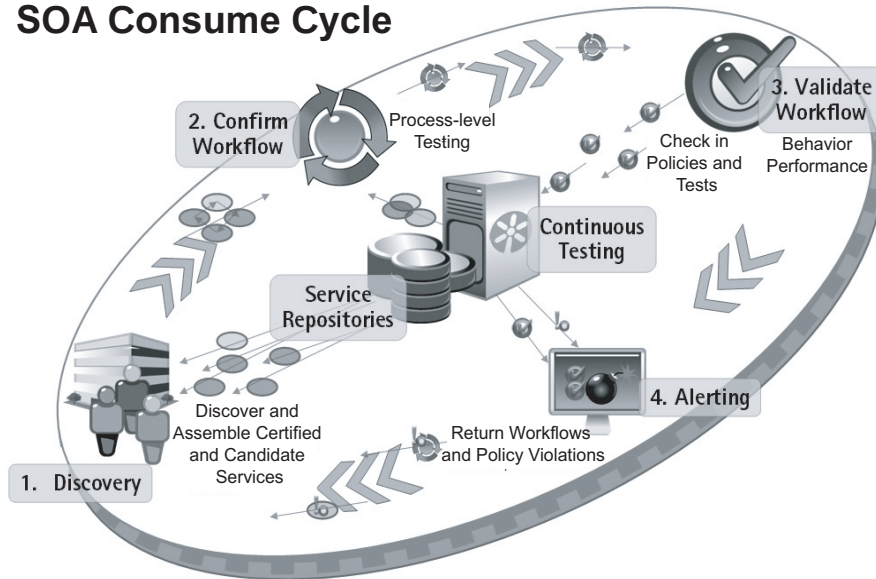


Figure 5: Consume Cycle for Sharing Testing and Certification Processes in a Net-Centric Environment

accelerate the process by submitting functional, performance, and other tests to validate the policy out of the box.

2. **Certify.** Certifiers use existing tests and iterate on those tests to validate that the SC meets all required policies at every stage of the service's life cycle. Tests are checked into the test registry/repository alongside the services, which are rated according to their level of certification (each organization defines its own certification levels, for example, from *trial level* of certification to *partially certified* to *fully certified*).
3. **Verify.** Initial certifications are not much good if they are not enforced later in production. Certifiers register the test cases, and they are run continuously to validate that expectations are met as the application environment evolves. Continuous verification happens on a regular time interval or based on any system level event.
4. **Review.** All metrics and test information is published to both certification and publishing teams for reporting and alerting on SC issues. Alerting mechanisms can inform any development or deployment team of exceptions or errors within the certification environment.

The Consume Cycle

The Consume cycle is equally important; those who plan to leverage published services must establish a model workflow outlining their expected behavior so that ongoing change does not cause the system to fail in unexpected ways (Figure 5).

How the Consume Cycle Works

Consumers browse the registry of available services and use them to define workflows that consume one or more SCs as steps needed to complete an objective.

1. **Discovery.** Development consumers browse available SCs and their associated published policies and test cases in order to determine applicability to their proposed workflows. Development teams can test them using existing tests or combined with a target workflow test (i.e., testing the validity of the workflow in absence of the underlying services).
2. **Confirm workflow and set policy.** Certifiers verify that the intended uses defined within the consumer's workflow are achievable, and once the workflow is certified, it is published to the registry as a set of expected behaviors – a policy – that can be certified via suites of test cases that accompany the workflow.
3. **Workflow validation.** Test suites for workflows are checked into a continuous testing process as policies for continuous monitoring of required quality of service – performance, scalability, and reliability. These tests set the bar for candidate and certified services that are accountable to support the workflow.
4. **Alerts and exceptions.** A test dashboard provides key metrics to development, certification, and administration teams. Workflows are monitored for issues as SC development life cycles and demands on deployment

evolve. If an exception, error, or boundary condition event occurs that violates one or more workflows, stakeholders can be alerted with root cause test cases provided.

Most organizations have not even considered managing policies or tests for how they consume services. There is no *free beer* in SOA. If there are no expectations placed upon the consumer of services, total chaos ensues, and there is no governance. By defining the expected behaviors of the consumer, service providers in the network can supply provisions for this, and validate that all the critical workflows are supported both now and in the future when each new release of the service component is proposed.

A Center of Excellence for SOA Life-Cycle Quality

Go Horizontal for SOA Excellence

One way to instill a sense of horizontal trust across the organization is through an SOA Center of Excellence (COE). For example, consider an analogy of states' rights vs. federal rights. The individual divisions (or states) need to consider themselves autonomous on certain levels and to owe certain rights at the federal level; but they also have the opportunity to participate in how policies are set at that federal level, so that it is not just a *down from above* edict that immediately creates a defensive reaction instead of a unified purpose.

Booz Allen Hamilton Vice President Art Fritzson noted the following:

I tend to reduce net-centricity to two questions that are both addressed by community. *Have you asked your community for help?* and *Are you helping your community?* If you get positive answers to both those questions, you've got 80-90 percent of what net-centricity promises just through behavioral changes. [4]

Which policies need to be tested as so-called federal policies and which can safely be handled at the state level? Policy testing can be automated along four domains: *structural*, *behavioral*, *performance*, and *runtime*. At the federal level, structural policy (or compliance) might be the most important aspect, while from a state-to-state, horizontal kind of policy, behavioral and performance aspects will perhaps be most important to define and test at those levels.

Publication and consumption of shared services will only happen in an environment where trust is fostered. Trust can never exist unless empirical data (test data) that supports assertions against expected policies about a component (service, component module, etc.) can be quantifiably and continuously obtained. The levels of certification, and what is defined as important to test, will differ for different communities of interest (COIs). Each federated consumer may reuse, or not reuse, those assets that will deliver efficiently for them.

Life-Cycle Quality and Testing

Software testing in the standard *waterfall* development cycle has long been relegated to a project milestone somewhere after development and integration happens. But in SOA, life-cycle quality is a continuous part of SOA governance and not an event that unit tests a specific technology as a pre-release *feel good*.

ZapThink's Ron Schmelter said the following on SOA quality:

Exposing a service is one step in the life cycle of SOA development, but it is not even the first step. Indeed, the step companies take to expose and execute Services should be one of the last they take as part of a mature architectural process. Quality, in particular, should be considered before any services are created. If a company has not considered how services will be tested, how consumers will reliably succeed or fail in their service consumption, and how they will iterate through service versions, then any consumers that bind to those initial services will be doing so at their own peril. [5]

The organization needs *complete* testing from the Web layer through Web services, databases, and all middle tiers of the application. Merely testing at a single layer will not uncover missed requirements.

Also, testing should support the entire extended set of players *collaborating* on SOA, from the process owners writing the requirements, to the developers implementing them in SOA, to the Quality Assurance teams verifying functionality and performance.

SOA testing should be continuous, not just in development and integration but in deployment because an SOA by nature is never a static application. Each

element of the SOA is on its own development and release life cycle.

The humble test is just like a bill sitting in Congress waiting to become a law (or an SOA policy). In order to get there, it is going to need the support and nurturing of the SOA community at a team level and across organizations.

In other words, we promote a test as an actionable aspect of SOA governance. To do this, the SOA test must contribute more than a specific technology checkpoint at a specific point in time. A test must span the SOA application continuously, and in so doing it becomes a verifiable SOA policy.

Conclusion

This article has outlined several key strategies for leading the way to a true net-centric approach to SOA life-cycle quality and testing, which is one of the three primary components of SOA governance.

Success in SOA is not something you can *buy* as a software package; it is something you must do. In fact, the quality of your policy and the relevance of your certification efforts depends entirely upon the skill and discipline level of all participants in the SOA strategy. The architect, the developer, the tester, and the requirements owner must work to establish trust, whether from a development perspective or a quality perspective.

The entire extended organization needs to adopt an SOA COE – the federal authority that helps the underlying *states* align around common goals. If we think about what the SOA COE must look like, then certainly there needs to be a set of participants that are not beholden to any particular one of the divisions involved, but there also needs to be significant membership from all of those who will be involved so that we get the participation. After all, this is not us vs. them – net-centricity is all of us in the same boat together. ♦

References

1. Linthicum, David. "Real World SOA." Sept. 27, 2006 <http://weblog.infoworld.com/realworldsoa/archives/2006/09/us_governmentso.html>.
2. Kastner, Peter. "The Composite Applications Benchmark." Aberdeen Group. Dec. 2006.
3. Kenney, L. Frank. "SOA Governance: It's More than Just Technology," Gartner, Inc. Nov. 2006.
4. Fritzson, Art. "Net-Centricity Survival Guide – Utilizing Communities of

Interest to Exploit Information Overload." Network Centric Warfare conference excerpt. Jan. 26-27, 2005, Washington, DC.

5. Schmelter, Ron. "Quantity Is No Measure of Maturity," ZapFlash report. Apr. 5, 2007.

Notes

1. ITIL stands for Information Technology Infrastructure Library, a set of best practices for delivering technology, used globally but largely originated in the United Kingdom. For more information see <www.itil.co.uk/>.
2. WSDL stands for Web Services Description Language, which is a protocol for identifying the properties of a directory or library of Web Services. SOAP stands for Simple Object Access Protocol, which is the common method (a form of XML) for transmitting data objects among Web Services and other technologies. For more, see <www.w3.org/TR/wsdl>.

About the Author



John Michelsen, is the founder and chief architect of iTKO, Inc. He has more than 15 years of experience as a technical leader at all organization levels, designing, developing, and managing large-scale, object-oriented solutions in traditional and network architectures. Michelsen is the chief architect of iTKO's LISA automated testing product and a leading industry advocate for software quality. Before forming iTKO, he was director of development at Trilogy Inc. and vice president of development at AGENCY.COM. Through work with clients such as Cendant Financial, Microsoft, American Airlines, Union Pacific, and Nielsen Market Research, Michelsen has deployed solutions using technologies from the mainframe to the handheld device.

iTKO Inc.
1505 LBJ FWY
STE 250
Dallas, TX 75234
Phone: (877) 289-4856
Fax: (817) 281-2458
E-mail: info@itko.com

WEB SITES

Service-Oriented Architectures (SOAs)

www.service-architecture.com

This site will help you get started with Web Services and SOAs. It features free articles, services, and product listings that can be used to develop an SOA using Web Services. There are nearly 400 pages of online articles covering many types of SOAs that provide an extensive overview of Web Services, related standards, and technologies that can be used in SOAs. Web Services make up a connection technology and is a way to connect services together.

Information Technology Library (ITL)

www.itl.nist.gov

ITL has been charged to lead the nation in utilizing existing and emerging IT to meet national priorities that reflect the country's broad-based social, economic, and political values and goals. Its extended charge continues under the Federal Information Security Management Act to develop cybersecurity standards, guidelines, and associated methods and techniques. Charged under other legislation, such as the U.S. Patriot Act and the Help America Vote Act, ITL is addressing the major challenges faced by the nation in the areas of homeland security and electronic voting.

Service Oriented Enterprise (SOE)

www.serviceoriented.org

ServiceOriented.org was developed to educate readers about the benefits and attributes of an SOE. ServiceOriented.org is part glossary, part educational story. The site does not have a rigid navigational structure, but instead invites readers to wander, following links to related topics as desired.

U.S. Army Enterprise Resource Planning Service-Oriented Architecture Resource Center

www.army.mil/escc/erp/soa.htm

As the Army embarks on transforming its warfighting capabilities, it is imperative that the business capabilities/enablers/processes transform to support the warfighter. Enterprise Resource Planning (ERP) systems provide an integrated suite of Information Technology applications that support the end-to-

end business operations of an entire organization. The ERP Resource Center is designed to provide enterprise process owners, program executive officers, program managers, and others involved in the business transformation of the Army with detailed information, supporting documents, and tools and techniques regarding the use of ERP systems.

The World Wide Web Consortium (W3C)

www.w3.org

The W3C develops interoperable technologies (specifications, guidelines, software, and tools) to lead the Web to its full potential. It is a forum for information, commerce, communication, and collective understanding. On this site, you will find W3C news, links to W3C technologies, and ways to get involved. New visitors can find help under the *Finding Your Way* heading.

SOA Magazine

www.soamag.com

The SOA Magazine is a monthly online publication dedicated to publishing specialized SOA articles, case studies, and papers by industry experts and professionals. Amidst all the SOA-related activity that is currently under way, there still remains a significant amount of confusion as to what exactly constitutes a SOA. Some qualify an SOA project by the fact that Web Services technologies are being used, while others classify SOA as a Web-centric variation of object-oriented design. The common criteria for contributions is that each explores a distinct aspect of service-oriented computing.

Wikipedia

http://en.wikipedia.org/wiki/service-oriented_architecture

This Wikipedia Web site provides links to multiple SOA Web sites, as well as information on SOAs. There is no widely agreed-upon definition of SOA other than its literal translation that it is an architecture that relies on service-orientation as its fundamental design principle. Service orientation describes an architecture that uses loosely coupled services to support the requirements of business processes and users. Resources on a network in a SOA environment are made available as independent services that can be accessed without knowledge of their underlying platform implementation.

MORE ONLINE FROM CROSSTALK

CROSSTALK is pleased to bring you this additional article with full text at <www.stsc.hill.af.mil/crosstalk/2007/09/index.html>.

SOA Security Reference Model

Nataraj Nagaratnam, Anthony Nadalin, Janet Mostow and
Sridhar Muppidi
IBM Software Group

Merely securing the perimeter with firewalls or routers is not sufficient for a flexible business. Security remains one of the biggest challenges given the complexity of a Service-Oriented Architecture (SOA) based environment, due to loose coupling of services and applications and their possible operations across trust

boundaries.

Security must be designed to cover the entire SOA environment, with services and applications capable of participating in the enterprise security via security services. In addition, security must be factored into the SOA life cycle, reflecting the fact that security is a business requirement, and not just a technology attribute. This article discusses a scenario and identifies a set of security requirements. It then talks about capabilities that can be used to address those needs.

Evolution in Action – Building Up to a Service-Oriented Architecture

Because this column has to be written several months in advance, I am writing it at the 2007 Systems and Software Technology Conference (SSTC) in Tampa Bay, Florida. Talk about a bunch of geeks! I mean this in a nice way, of course (because I know I *am* one).

Seriously, the SSTC had some great exhibits this year. Most of the exhibits, of course, displayed software products designed to help you develop and maintain your systems, which leads me to the topic of this column: Service-Oriented Architectures (SOAs). SOAs are loosely defined as *an environment made available as independent services that can be accessed without knowledge of their underlying platform implementation*². Interoperability – what a concept. You see, it's all evolutionary, because....

... IN THE BEGINNING was machine language. You remember?

BALR R14, R15 USING *,**

The problem was that machine language was so closely tied to the machine that it wasn't even transportable across different machines from the same vendor. Thus, programs that ran on an IBM 1401 with a specific memory configuration would not even transport to another differently configured 1401. What we needed was...

... HIGH-LEVEL LANGUAGES. FORTRAN, COBOL, RPG, and eventually, languages like C and Standardized are transportable (within limits) from one machine to another. Which leads to....

... PARAMETER PROBLEMS. If a C program passed two parameters (such as an integer and a real), but the receiving sub-program read them as a real followed by an integer, it would try and work and usually fail because the data was interpreted incorrectly. When I taught at the U.S. Air Force Academy back in the '80s, we used Pascal and always taught that parameters had to agree (between caller and callee) in number, order, and type. Early on, software engineers found out that about 75 percent of all errors occurred not directly in the code but in the code interfaces. So we tried to emphasize to students to *always* check the number, order, and type of parameters. Of course, telling them to check their parameters was not as good as ...

... ENFORCED PARAMETERS AND STRONGLY TYPED LANGUAGES, Ada (and its successors), C++ (to an extent), and Java. Want to pass four parameters that consist of two reals and two floats? Then let the compiler and run-time environment *enforce* the parameter number, order, and type. If you tried passing them in the wrong order, then the program would return an error and not attempt to convert incorrect data. In fact, if you accidentally try and pass a floating point number as an integer (and perhaps lose precision), the compiler/runtime environment will also prevent that, giving you a better chance at correct programs and preventing accidental parameter type mismatches. However, as programs grew larger and larger (and systems of systems evolved) the interfaces between multiple sub-programs became larger and larger (with more and more parameters), which leads to...

... STANDARDIZED LIBRARIES. Why bother to pass a

zillion parameters to a handwritten user display procedure (which took lots of time to write), when a completely pre-written Graphical User Interface was available? All you have to do is use an object-oriented language and development environment, spend a little time researching which services and libraries are available, and then inherit/instantiate the code you need. Don't write it – REUSE IT! However, to make standardized libraries and reusable code/services efficient and cheap, we really needed...

... STANDARDIZED OPERATING SYSTEM SERVICES. Back in the '80s and '90s, you couldn't trust the operating system (OS) code. You used the OS to load your program, but then you wanted to write your own memory management, real-time scheduling, and other critical services. However, as OSs became more and more standardized, they provided more and more services. In fact, what you wanted was a reasonably secure and reliable set of services that would let you not just reuse code, but also let you utilize reusable services. Which is why we need...

... SERVICE-ORIENTED ARCHITECTURES. It's all about saving time and money. High-level languages are cheaper than assembly language. Standardized OSs give you services that are cheaper than *roll your own*. Reusing code saves you time and money. So why not have take advantage of *independent services* with defined interfaces that can be called to perform their tasks in a standard way, without the service having foreknowledge of the calling application, and without the application having or needing knowledge of how the service actually performs its tasks. SOAs are an evolutionary step above standardized OS services, and can be regarded as *a style of information systems architecture that enables the creation of applications that are built by combining loosely coupled and interoperable services*². All of the good software engineering buzzwords that we have tried to teach over the last 20 or so years are inherent in an SOA: loosely coupled, interoperable, abstractions, and enforced interfaces.

There you have it – engineering evolution in a nutshell. By using an SOA, you will achieve savings in both development cost and time. As a single example, interprocess communication and network interface issues will have been solved in an SOA (and when they need updating, the SOA will be updated, not your program!). It's just like plagiarism but without the moral dilemma.

Why do you want to reinvent the wheel when there are already perfectly good wheels that somebody else has already built?

— David A. Cook, Ph.D.

The AEGIS Technologies Group, Inc.
dcook@aegistg.com

Notes

1. Well, where else could you find an audience that appreciates the following? “Obviously, God LOVES the C programming language, because in Genesis 1, verse 2, it clearly states that the earth was without form, and it was VOID.” And, for you FORTRAN programmers: “God is REAL (unless explicitly declared INTEGER).”
2. Wikipedia <http://en.wikipedia.org/wiki/service-oriented_architecture>.



Homeland
Security

Software Assurance Program

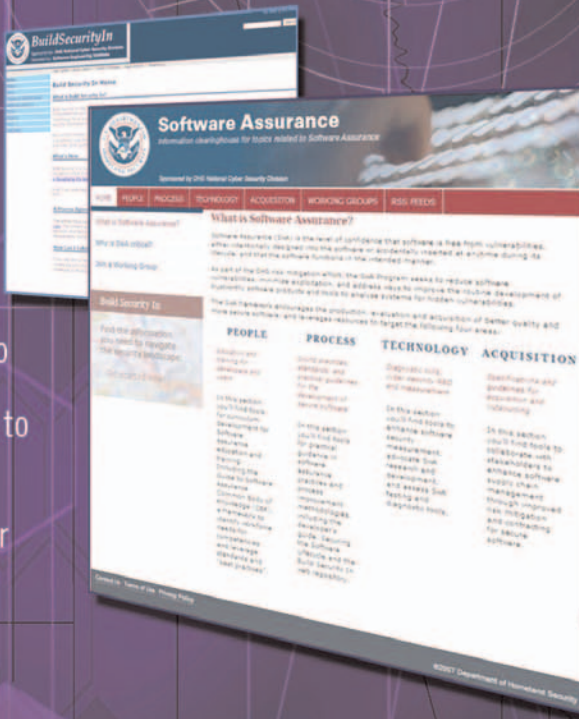
Software is essential to enabling the nation's critical infrastructure. To ensure the integrity of that infrastructure, the software that controls and operates it must be reliable and secure.

Visit <http://www.us-cert.gov/SwA> to learn more about the software assurance program and how you can become more involved.

Security must be "built in" and supported throughout the lifecycle.

Visit <http://BuildSecurityIn.us-cert.gov> to learn more about the practices for developing and delivering software to provide the requisite assurance.

Sign up to become a free subscriber and receive notices of updates.



<http://www.us-cert.gov/SwA>

The Department of Homeland Security provides the public-private framework for shifting the paradigm from "patch management" to "software assurance."

CROSSTALK / 517 SMXS/MXDEA

6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

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

CROSSTALK is
co-sponsored by the
following organizations:



NAV AIR



Homeland
Security