SYS-CON

THE WORLD'S LEADING MAGAZINE DEDICATED TO WEB SERVICES TECHNOLOGIES

FOR ALL THE LATEST NEWS AND INFORMATION VISIT:
www.SOA.SYS-CON.com

SOAWORLD™ MAGAZINE

JULY 2007 / VOLUME: 7 ISSUE 7

The 3 Faces of SOA

8 RUSSELL LEVINE

PLEASE DISPLAY UNTIL SEPT. 30, 2007

$6.99US $7.99CAN

07>

0 71486 03420 9

SEE PAGE 23

_INFRASTRUCTURE LOG

_DAY 68: The business climate is constantly changing. Our IT environment is completely rigid. We can't align IT to meet the larger business needs. I told Gil we need an SOA so we can be proactive for once.

_DAY 70: Gil had an idea. He calls it a GOA (Gil Oriented Architecture). He brought in a bunch of contractors over the weekend and made the entire office "modular" and "flexible."

_Gil says I'm looking at the new standard in architecture. I say I'm looking at a giant habitrail. We need help.

Plays Well with Others

WRITTEN BY **SEAN RHODY**

Recently SOA World Magazine was the host of a conference on SOA and Web 2.0 in New York City. SOA World 2007 brought together an amazing group of IT professionals who helped describe and expand the definitions of SOA.

One of the most interesting challenges faced by IT is the role of Web 2.0 in service-oriented architecture. This is caused in part by the fact that you can do Web 2.0 without service-oriented architecture, and you can also do service-oriented architecture without Web 2.0. And yet, the key question becomes should you or will you?

I've long gone on record as saying we need something better than the browser to truly deliver zero footprint Internet-based applications. We need an independent platform specification accompanied by platform-specific implementations that allow for the best of all worlds, not the least. The browser must die (I've said this before), and from its ashes we need a true application platform.

That being said, Web 2.0 is a step in the right direction. AJAX adds asynchronous communication and frees us from the dreaded refresh tag. That's not the only thing AJAX provides, but that alone makes the world a better place. We've long needed a way to get data back to the screen without having to repaint it. Seems simple, but look how long it took.

Web 2.0 is more than just AJAX. RSS feeds and blogs provide new ways to publish and edit Internet-based content and form communities. Flash and a host of other technologies are going a long way to provide the rich Internet application landscape that may make the browser truly useable instead of downright annoying. In time, the plug-ins may replace the browser entirely, which would be a godsend (in my opinion at least).

Yet none of these things is inherently service based. Or, rather, they don't have to be used that way. They can be combined with today's typical application style – that of a silo application that has no concept of service – without penalty or fear. Whether that's the best approach is not really the issue.

SOA does not come with the concept of a user interface, at least by most definitions. Services are defined, producers produce, and consumers consume, but the concept of actually defining a human interaction layer seems to get lost in the shuffle. As important as SOA has become in rescuing the enterprise from the failure of EAI and allowing computer-to-computer communication across boundaries, the average SOA proponent tends to view the human user as unimportant. Like our text books used to say, "That's left as an exercise for the reader."

That's a problem. SOA is not a god, and while IT is right to move to it for many things, it's important to realize that every service we build is going to ultimately be used for the benefit of some end user. The lack of a user interaction layer within SOA is a flaw that may ultimately limit the applicability of SOA as an enterprise architecture approach.

We need a human / user interaction layer as a core concept within the SOA technology stack. As tempting as it is to relegate that to the actual implementers and pass on it as too complex a task to standardize, it's exactly that complexity that cries out for some common definition and rigor.

We need the vendors who are bringing SOA to the market to coordinate with the Web 2.0 community. We need the intersection of service and interface, that place in the middle where SOA includes a user interface and Web 2.0 defines itself as a service consumer (and also a provider of services within the user interface layer, since the concept of service can be expressed at multiple levels). This issue addresses those challenges and provides some insight into making these two technologies co-exist. ■

About the Author

Sean Rhody is the editor-in-chief of **SOA World Magazine**. *He is a respected industry expert and a consultant with a leading consulting services company.* *sean@sys-con.com*

SOA WORLD MAGAZINE
www.SOA.SYS-CON.com

EDITORIAL OFFICES
SYS-CON MEDIA
577 CHESTNUT RIDGE ROAD, WOODCLIFF LAKE, NJ 07677
TELEPHONE: 201 802-3000 FAX: 201 782-9637
SOA World Magazine (ISSN# 1535-6906)
Is published monthly (12 times a year)
By SYS-CON Publications, Inc.
Periodicals postage pending
Woodcliff Lake, NJ 07677 and additional mailing offices
POSTMASTER: Send address changes to:
SOA World Magazine, SYS-CON Publications, Inc.
577 Chestnut Ridge Road, Woodcliff Lake, NJ 07677

The Scaling Crisis Around SOA

Scalable SOA solutions are emerging

WRITTEN BY **DAVID S. LINTHICUM**

Making solutions scale is nothing new. However, the SOA technology and approaches employed recently are largely untested with higher application and information and service management traffic loads. SOA implementers are happy just to get their solutions up and running, but, in many cases, scalability has simply not been a consideration with SOA, nor is load testing, or other performance fundamentals for that matter. We're seeing the results of this neglect now that SOA problem domains are exceeding the capacity of their architectures and the technology in many instances.

My daily routine is to answer e-mail from somebody, somewhere, asking me why his SOA doesn't scale. Unfortunately, the answer is something these people don't want to hear. It's really the fact that they took the wrong approach, used the wrong technology, and the fix is going to be painful. However, you can avoid these issues just by doing some additional planning and testing upfront before you commit to a solution.

The core issue is that many SOA technology vendors haven't focused on scalability in their solutions. Instead, feature/function enhancements are the rule of the day. It's more important to add orchestration features and more adapters to your solution than to figure out how to pump more information, and manage more services. So these single-threaded solutions, on top of the issues around Web Services in general, make for solutions that are more about integration than true business transaction loads. Not to mention supporting the notion of both reuse and agility.

Its dependence on the traditional architectures is the core problem of scaling SOA solutions. The most popular SOA technologies require that all information and services under management do so in a single server domain (in most cases). This processing is a mixture of service abstraction, service management, schema and content transformation, rules processing, message splitting and combining, as well as message routing (ESBs). This is despite the fact that Web Services, at their essence, are more about the distributed service invocation model that's not as well followed when considering instances of technology. Indeed, most SOA solutions out there are bound to a single service approach, and don't provide a smooth path to scalability.

While the scaling crisis around SOA isn't well known at this point, considering the fact that the larger SOA projects have just started moving, there are a few vendors that I've been watching that provide the distributed architecture needed to bring better distributed reliability to SOA. Rogue Wave's HydraSCA, for instance, provides a service grid based on "Software Pipelines," an architecture and methodology that enables the development and deployment of scaleable SOA-based applications. Rogue Wave is promoting Software Pipe lines as a cross-vendor approach to service distribution, and other key players are in there as well using similar implementation patterns.

What's interesting is that Software Pipelines, the approach and the instance of Hydra technology, increases scalability by providing concurrent computing of business services in and across servers, while preserving business rules.

—continued on page 12

About the Author

David S. Linthicum is an internationally known application integration and Service Oriented Architecture expert. In his career Dave has assisted in the formation of many of the ideas behind modern distributed computing including Enterprise Application Integration, B2B Application Integration, and Service Oriented Architecture, approaches and technologies in wide use today. Currently, he is CEO of the Linthicum Group, LLC, (www.linthicumgroup.com) a consulting organization dedicated to excellence in Service Oriented Architecture planning, implementation, and strategy.

david@linthicumgroup.com

SOA WORLD MAGAZINE
www.SOA.SYS-CON.com

CORPORATE

President and CEO
Fuat Kircaali fuat@sys-con.com

Group Publisher
Roger Strukhoff roger@sys-con.com

ADVERTISING

Senior VP, Sales & Marketing
Carmen Gonzalez carmen@sys-con.com

Advertising Sales Director
Megan Mussa megan@sys-con.com

Associate Sales Manager
Corinna Melcon corinna@sys-con.com

SYS-CON EVENTS

Event Manager
Lauren Orsi lauren@sys-con.com

Event Associate
Sharmonique Shade sharmonique@sys-con.com

CUSTOMER RELATIONS

Circulation Service Coordinators
Edna Earle Russell edna@sys-con.com
Alicia Nolan alicia@sys-con.com

SYS-CON.COM

VP information systems
Bruno Y. Decaudin bruno@sys-con.com

Consultant information systems
Robert Diamond robert@sys-con.com

Web Designers
Stephen Kilmurray stephen@sys-con.com
Richard Walter richard@sys-con.com

ACCOUNTING

Financial Analyst
Joan LaRose joan@sys-con.com

Accounts Payable
Betty White betty@sys-con.com

SUBSCRIPTIONS

SUBSCRIBE@SYS-CON.COM
1-201-802-3012 or 1-888-303-5282
For subscriptions and requests for bulk orders,
please send your letters to Subscription Department
Cover Price: $6.99/issue
Domestic: $69.99/yr (12 issues)
Canada/Mexico: $89.99/yr
All other countries: $99.99/yr
(U.S. Banks or Money Orders)

Worldwide Newsstand Distribution:
Curtis Circulation Company, New Milford, NJ

For list rental information:
Kevin Collopy: 845 731-2684, kevin.collopy@edithroman.com;
Frank Cipolla: 845 731-3832, frank.cipolla@epostdirect.com

- **Immerse yourself in SOA** instead of dealing with tools that tack SOA features onto legacy tooling. With the Toolkit you don't have to deal with the impedance mismatch between your development habitat and the SOA paradigm because none exists.
- **Define and update** Web service interfaces in a quick, collaborative manner. Each time an interface is introduced or revised, a prototype implementation becomes available at the click of a button.
- **Generate XML** documents for requirements or testing by building intelligence into the document definition itself, what we call an "executable" document. No more dealing with blunt instruments like templates or DOM.

NEXT GENERATION WEB SERVICE TOOLING

Announcing The Delve® SOA Fast-Prototyping Toolkit

Web service prototyping, XML document generation, and scripted service testing, all with the world first commercial service-oriented programming language.

Delve® Integrates a code editor, run-time environment, specialized Web server and SOAP stack in a single, simple application.

Supports SOAP/XPath/XML-Schema/XML

CretaceousSoftware™
Evolving SOA

Download the Delve® SOA Fast-Prototyping Toolkit (Beta1.0)

www.cretaceoussoftware.com

The Three Faces of SOA

COTS, Legacy, and New Development

WRITTEN BY **RUSSELL LEVINE**

Business Takeaways

- Service-enabling strategies for application integration depend on whether the applications to be integrated are purchased, previously developed in house, or being built from scratch.
- A service oriented architecture (SOA) is as relevant for application integration as it is for application construction.

Technology Takeaways

- Sometimes an Enterprise Service Bus (ESB) is the right approach to provide SOA features.
- Sometimes invasive application modification is the right approach to provide SOA features.

You have purchased applications. You have existing in-house applications. You have applications you are in the process of writing from scratch. Now your CIO wants to know how all these applications are going to start leveraging this "SOA" that's been in all the papers.

Ah yes, S-O-A, the elusive Service Oriented Architecture. You've read the analyst reports. You've watched some online webinars. You even have a fancy poster in your office. It all sounds good, but you're not quite sure how it applies to your environment.

The bottom line is:

1. SOA is relevant even if you're not focused strictly on building applications.
2. You need the right infrastructure.
3. You must consider the provenance of your applications to take the best approach to enable them for SOA.

The remainder of this article will illustrate how the application of these basic concepts can help you evolve your current IT landscape to one that takes advantage of the opportunities offered by an SOA. But before we get too deep into the details, let's clarify what we're referring to as an SOA and look at how this can be interpreted to maximize its benefits.

SOA OK

An SOA is typically described as an approach to building applications based on well-defined, loosely coupled processes or "services." This design approach has been appreciated as a theoretical

model for years, but the advent of standards for Web Services has recently allowed SOA to gain a significant foothold in the industry. A full discussion of SOA and Web Services standards is beyond the scope of this article. Interested readers are encouraged to review back issues of this publication for a variety of pieces on these subjects. For now, we will merely point out that Web Service standards enable SOA in a manner which is language-neutral, platform-independent, and based on well-established Internet standards.

To Serve IT

SOA and Web Services have historically been discussed in the context of software development, when an application is built from the ground up. Application development is also where SOA and Web Services have gained real traction, for example, with software vendors and IT shops where business requirements are met with custom applications.

Today, however, many IT shops no longer develop software as a core competency. New capability is primarily provided via the acquisition of commercial off-the-shelf (COTS) software or by modifying previously developed applications. These previously developed legacy applications either pre-date the adoption of a COTS strategy or were deployed to achieve some competitive advantage.

The value these shops deliver in the provisioning of new functionality therefore centers on the integration of independently designed COTS, legacy, and occasionally newly developed applications. Differentiation is driven by this mix and by how well these systems work together. In such shops it may not be obvious how this newfangled SOA applies.

However, as some integration architects have recognized, SOA leverages the same principles and offers the same advantages as those associated with Enterprise Application Integration (EAI) approaches. The basic difference between EAI and SOA is that SOA is usually discussed in terms of a particular application whereas EAI applies at the application portfolio level. For example, the loosely coupled SOA approach to application assembly is analogous to the loosely coupled EAI approach to application integration.

What's All This Then?

It all sounds very interesting but what does any of this mean in practice for "buy and integrate" IT shops? What implication does the SOA model have for these organizations? Do Web Services play a role in meeting day-to-day requirements?

One way to answer these questions is to consider the roles systems can play in an SOA and the application model they follow.

The basic roles in an SOA are the service provider and the service consumer. The first role is taken by the application that presents services. The second role is taken by the application that exercises services to accomplish some business objective. In reality, services are often combined or composed into more complex processes, but conclusions drawn from the simpler interactions can be extrapolated to these more sophisticated scenarios.

The application models — new custom applications, legacy, and COTS — should be familiar to most readers. For the uninitiated, the sidebar provides a brief primer.

For service-based integration to be viable service producers must possess a service presentation capability and they must present services relevant to integration requirements. Service consumers must have a service consumption capability and must be able to invoke services at points relevant to integration requirements in a manner supported by the available services.

One other consideration is an integration infrastructure, specifically what is commonly referred to as an ESB, described in the sidebar.

For each application model, we recommend SOA enablement strategies in the context of service provider and service consumer roles. Aside from using native service capabilities, two basic approaches are discussed: applying ESB technology and invasive application modification.

Table 1 summarizes the recommendations for COTS, legacy, and custom development scenarios using a familiar stoplight metaphor. These are the three faces of SOA.

COTS – It Is What It Is

Ultimately, whether a COTS application is service-enabled is at the discretion of the vendor. Note that most vendors are taking an SOA approach to the construction of these applications because of the advantages of this approach from an application build standpoint. Of course, for many implementation teams, the internal COTS architecture is obscured inside a black box. So this application of SOA in and of itself has little impact.

However, when SOA is the internal build approach, it's also likely to be an external integration approach. That is, service-based interfaces, usually following Web Services standards, are likely to be available, at least as an option.

The only real point of influence customers have therefore is the extent to which these capabilities are considered in purchase

Application Models Monograph

The "new custom applications" model refers to so-called "green field" applications where no such functionality currently exists and an application is being built to specification. A legacy application model can be thought of as "brown field" development. This is where existing applications are modified to meet business requirements. The current application constrains possible approaches much the same way that a mothballed industrial site constrains future land use options. The COTS model, of course, refers to the purchase of an application from a vendor.

Can I Buy Your Service Bus?

An Enterprise Service Bus (ESB) is a key infrastructure element in an SOA implementation supporting application integration. The most relevant ESB feature from an SOA perspective is the ability to broker the various legacy protocols of existing applications, the proprietary protocols of COTS applications and the Web-based protocols of services-enabled applications. An ESB also typically provides other capabilities such as process orchestration, message transformation and routing, state management and the mediation of different integration patterns. These and other functions should be familiar to most integration architects.

	ESB Services Enablement	Invasive Services Feature Creation
COTS	Yes	No
Legacy	Sometimes	Sometimes
Custom	No	Yes

Table 1: THE THREE FACES OF SOA

decisions. Ironically, even this influence is limited because COTS application selection is usually based largely on business priorities.

The Service Provider Role

Where integration requirements can be supported by COTS services and the applications to be integrated are capable of consuming such services, they should do so. Service-based interfaces are likely to represent the most robust, longest-lived interface approach. The loosely coupled nature of these interfaces will also make integration based on these interfaces more flexible in the face of application upgrades to either the COTS application or the applications to which it is integrated.

Where a COTS application has relevant integration points but does not have a service presentation capability, an ESB may allow a COTS application to still play the role of a service provider. Where a COTS application doesn't have relevant outbound integration points, you must compromise requirements, develop a workaround, or take on the unsavory task of modifying the COTS code. This last option can be realized in-house, by the vendor, or even a third party, but it defeats some of the advantages of a COTS strategy and many IT organizations are therefore reticent to pursue it.

The Service Consumer Role

The consumption of services by a COTS application will hinge primarily on two factors. First, does the vendor provide the capability to exercise services at the required integration points? Second, do these services exist in the applications to which the COTS application needs to integrate? As in the service provider scenario, where these stars align, a services-based approach is a good strategy. Moreover, to the extent that these integration requirements are present and the COTS applications can consume services, this can actually drive build requirements for new development or legacy modifications. That is, if you are going to build or modify an interface anyway, you might as well present it as a service when the calling application can consume it as such.

When the COTS application has a relevant integration point but does not have a service consumption capability, you can consider an ESB service consumption strategy. For example, many COTS applications have proprietary inbound interfaces — say, based on interface tables — that can be managed via an external agent such as an ESB.

When a COTS application doesn't have needed inbound integration points, these requirements can be addressed with approaches similar to those employed when a COTS application doesn't have desired outbound integration points.

Legacy Applications – Wrapper's Delight?

The fact of the matter is, it's often not any easier to meet legacy integration requirements with services than it is with any other integration approach. Sometimes we are considering applications whose designers never contemplated such requirements. The fundamental architecture of the application may impose obstacles to integration based on underlying data models or processing assumptions. Or, there can be an impedance mismatch with the typical service-based request-reply integration pattern and the native integration mechanism of the application.

This is why the frequently offered suggestion that you can just "wrap" legacy systems with services and be on your way is a hoax.

> **"For presenting services, you need, first and foremost, to expose the appropriate integration point"**

The Service Provider Role

For presenting services, you need, first and foremost, to expose the appropriate integration point. To the extent that existing systems have object-oriented foundations or their designers considered similar requirements, you may have an easier time of it; that is, if you happen to have the appropriate interface already exposed in a service-like manner, converting it to conform to Web Service standards may not be too challenging. You can even leverage an ESB to do much of the work. However, where this is not the case, plenty of heavy lifting may be required.

Unlike a COTS scenario though, it's perfectly reasonable to take an invasive approach to add service-based interfaces to legacy applications. That is, modifying the application to provide relevant integration points may be perfectly feasible. It's just important to recognize that building services is a more significant undertaking than simply service-enabling existing outbound interfaces.

The Service Consumer Role

As far as legacy applications invoking services go, there are three considerations. First, this approach is only worth considering where relevant services are present. Then, you need to consider the capability of the legacy application to technically consume these services, again typically Web Services. This capability is often based on the development toolset and/or runtime environment of the legacy application. Finally, you have the same basic application architecture issues involved with presenting services. That is, can you crack open the legacy application in such a manner that it can take advantage of any services that might be available?

When relevant legacy consumption integration points exist, an ESB can often be layered on top of them to invoke services where they are present.

If these integration points aren't present, as with service provisioning, you may need to modify the application directly.

New Development – The Final Frontier

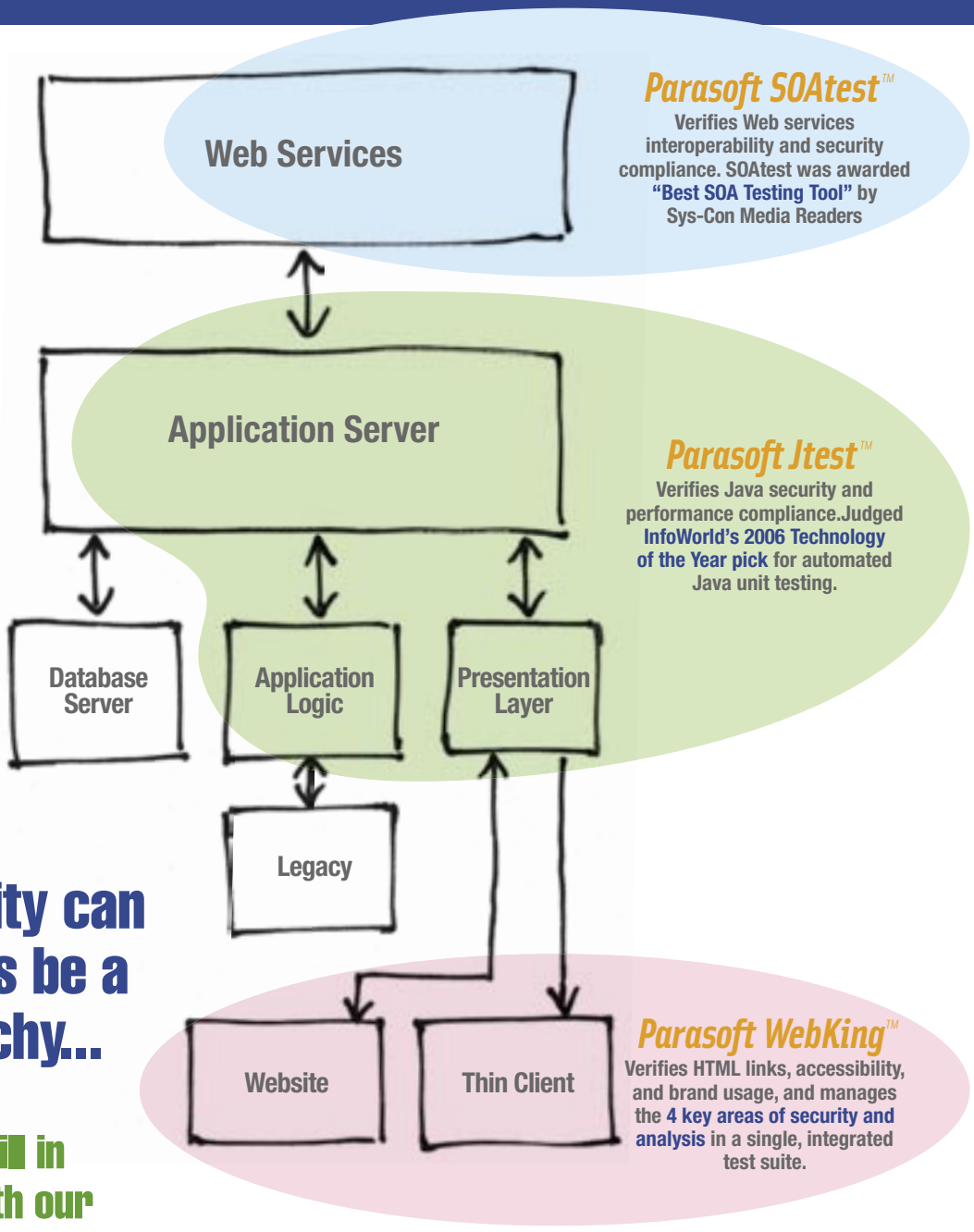
While new development may be increasingly rare in many IT shops, when it occurs, you have ultimate control since these applications are effectively "under construction" as requirements are captured.

The Service Provider Role

The recommendation for service presentation would be to take the same approach that most COTS vendors are taking — use Web Services. One caveat though is that while IT professionals are adept at abstracting and generalizing functionality for known requirements, you can't predictably build reuse beyond this point because any additional requirements are, well, unknown. No technology or architecture will address this issue. Technology like those based on Web Services standards will allow for integration to unanticipated platforms, but not in unanticipated patterns or for unanticipated purposes.

The Service Consumer Role

The extent to which new applications can consume Web Services



Parasoft SOAtest™
Verifies Web services interoperability and security compliance. SOAtest was awarded **"Best SOA Testing Tool"** by Sys-Con Media Readers

Parasoft Jtest™
Verifies Java security and performance compliance.Judged **InfoWorld's 2006 Technology of the Year pick** for automated Java unit testing.

Parasoft WebKing™
Verifies HTML links, accessibility, and brand usage, and manages the **4 key areas of security and analysis** in a single, integrated test suite.

Improving productivity can sometimes be a little sketchy...

Let Parasoft fill in the blanks with our Web productivity suite.

Parasoft products have been helping software developers improve productivity for over 20 years.

Jtest, WebKing, and SOAtest work together to give you a comprehensive look at the code you've written, so you can be sure you're building to spec, the new code doesn't break working product, and any problems can be fixed immediately. Which means you'll be writing better code faster.

So make Parasoft part of how you work today. And draw on our expertise.

PARASOFT®
We make software work.™

Go to www.parasoft.com/WSJmagazine • Or call (888) 305-0041, x3501

will make them easier to integrate as time marches on. This is because the integration mechanisms of the systems to which these new applications will need to integrate should be moving towards Web Services standards themselves. So where complementary services exist, this should be the integration strategy. Moreover, even if the applications with which you need to integrate use non-service-based approaches, it can still make sense to build service-based interfaces and then implement them using an ESB. That way, you have a migration path towards a future, more fully service-enabled integration world.

Again, this advantage is only at the platform level, not the application behavior level. Still, the good news is that since you're building from scratch, just as with the presentation of services, you can insert the application integration points in the appropriate location and craft them to follow appropriate integration patterns.

Also, while you don't need an ESB to service-enable integration points that you are building as services, you may still want to leverage other useful ESB features such as orchestration or processes monitoring. By the way, this last point applies to COTS and legacy application models as well.

I Hear New Trends a Comin', They're Rollin' Round the Bend

One final note on an ESB is that it also supports a co-existence strategy along the road to full application portfolio service-enablement. That is, you are never going to be able to "flip the switch" on an SOA for all your applications en masse. So you're going to need a way for your new service-enabled applications to co-exist with the traditional integration architectures of applications waiting in the wings for their service-enablement makeover.

Besides, even though this may be a noble goal, it may not be practicable or justified. Therefore a co-existence strategy is going to be a requirement for the foreseeable future, especially because as far as integration innovations go, this is just the latest. As soon as you get close to your SOA utopia, the target will move. Some other clever strategy will be bandied about there promising the next quantum leap in IT effectiveness. An ESB can then provide a bridge between your SOA-based application set and whatever comes next.

Let's Go Servin' Now, Everybody's Learning How

These are the three faces of SOA: COTS, legacy, and new development. By recognizing these faces and the approaches they imply, you can deliver a more effective SOA. You can elevate your SOA from an application construction tactic to an enterprise application integration strategy, with a commensurate elevation in returns. And in the end, that's probably what your CIO wants anyway. ■

About the Author

Russell Levine is an architect with the Unisys Business Enabling Technology Team. He provides technical leadership for integration-related internal Unisys projects and professional services engagements. Russell has over 20 years of experience in IT applications architecture, development, support and consulting. He holds an MBA from the University of Michigan.

russell.levine@unisys.com

The Scaling Crisis Around SOA

—continued from page 6

For those of us who have been in the business for a while, this is common sense scalability, meaning that the processing is distributed across more than one processing entity, and so the throughput increases using this parallel processing model. As long as this distribution is managed well...it's very effective.

There are other SOA solutions moving in this direction as well, and as SOA becomes more accepted, the ability to make your SOA scale is clearly going to be on the critical path. What's key is that those who select SOA technology that needs to scale consider the following basic principles:

- Consider performance and scalability as something that's systemic to the architecture; it can't be an afterthought.
- Make sure to do a proof-of-concept to demonstrate that the technology works as advertised, and work with the vendor to understand the best way to implement the technology.
- Consider operational issues during the design. Who's going to maintain it, and how?
- Learn how to do basic performance and scalability modeling. While never perfect, they do indeed provide you with a jumping off point when considering high performance and scalable architectures.
- Never be afraid to change approaches and technologies if they don't seem to be working out. ■

SOA and Service Virtualization

—continued from page 27

Standalone service containers aren't enough. Large-scale adoption of SOA means services distributed throughout the organization on different machines and even different platforms. These services must run on an interconnected grid of service containers. Service invocations must flow seamlessly from one "node" in this grid to any other node. In fact, it shouldn't matter which machine in the grid a given service is deployed to because the service code itself doesn't contain any hardwired ties. Invoked services might live on the machine from which they are called, or any other machine on the grid.

This extreme decoupling has radical implications for service scaling. Just as machine virtualization makes it easier to provision new machines, service virtualization makes it easier to provision new services. In the SOA world, the ability to provision new instances of a specific service is very important. The web of reuse that one sees in mature SOA makes anticipating load requirements nearly impossible. The ability to react to those load requirements by provisioning new services makes predicting them unnecessary.

Large buildings rely on a single solid foundation and a well-designed infrastructure to survive. Large-scale SOA is no different. Building your services on top of a solid service virtualization layer will reduce operational cost and complexity while allowing your developers to focus on writing new functionality and providing new business value. In today's hyper-competitive business climate this increased agility provides an invaluable edge. ■

About the Author

Rourke McNamara is senior product manager, products and technology, at TIBCO Software, Inc., responsible for supporting the company's service-oriented architecture technologies. He has also worked within the Professional Services Group at TIBCO as senior software architect. Prior to joining TIBCO, he was a senior consultant with Gemini Systems and director, Product Development and Systems, for IC Group. Rourke received his BS in computer science from the University of Pennsylvania.

Why is the WSO2 ESB the hottest ESB out there?

... its two products in one!

Full featured ESB

- Proxy services - transport (HTTP/S, JMS, SMTP), interface (WSDL/Schema/Policy), message format (SOAP/POX), QoS (WS-Security/RM) and optimization switching (MTOM/SwA)
- Integrated Registry/Repository, facilitating dynamic updating and reloading of configuration and resources
- Highly flexible, supports XSLT, XPath, Java, Ruby, Javascript

High performance XML Gateway

- Load-balancing/fail-over and throttling support
- Enforce WS-Security through policies, or SSL offloading for SOAP
- Validate, transform and route messages
- Non-blocking http/s transport for ultrafast execution and support for a large number of connections

WSO2 Enterprise Service Bus

Simple to get started and use

- Ships with many end-to-end samples that work out-of-the-box and comes complete with sample client and service code

Web based administration

- Easy to configure even complex standards like WS-ReliableMessaging through simple check boxes

Full XML support

- Built-in support for XML, Namespaces, XPath, XSLT and XOP

Support for non-XML messages

- Supports binary messages, as well as pure text or binary JMS messages etc.

Extensible

- By using Java/Spring classes or BSF scripting languages such as Javascript, Ruby, Groovy etc.

Multi protocol

- Java NIO based non-blocking http/s, JMS, Mail, File Apache VFS files systems such as - S/FTP, Zip, gzip, local, http/s..

WSO2
Oxygenating The Web Service Platform

http://wos2.com

Approaching SOA Testing

Even testing needs testing

WRITTEN BY **DAVID S. LINTHICUM**

So, does testing change with SOA? You bet it does. Unless you're willing to act now, you may find yourself behind the curve as SOA becomes systemic to all that is enterprise architecture, and we add more complexity to get to an agile and reusable state.

If you're willing to take the risk, the return on your SOA investment will come back three fold...that is, if it is a well-tested SOA. Untested SOA could cost you millions.

Truth be told, testing SOAs is a complex, disturbed computing problem. You have to learn how to isolate, check, and integrate, assuring that things work at the service, persistence, and process layers. The foundation of SOA testing is to select the right tool for the job, having a well-thought-out plan, and spare no expense in testing cycles or else risk that your SOA will lay an egg, and have no creditability.

Organizations are beginning to roll out their first instances of SOA, typically as smaller projects. While many are working fine, some are not living up to expectations due to quality issues that could have been prevented with adequate testing. You need to take these lessons, hard learned by others, and make sure that testing is on your priority list if you're diving into SOA.

How Do You Test Architecture?

The answer is, you don't. Instead you learn how to break the architecture down to its component parts, working from the most primitive to the most sophisticated, testing each component then the integration of the holistic architecture. In other words, you have to divide the architecture up into domains, such as services, security, governance, etc. and test each domain using whatever approach and tools are indicated. If this sounds complex, it is. Indeed, the notion of SOA is loosely coupled complex interdependence and so the approach for testing must follow the same patterns.

Before we can properly approach SOA testing, it's best to first understand the nature of the concept of SOA, and its component parts. There are many other references about the notion of SOA, so I won't dwell on it here. However, it's the foundation of the approaches and techniques you'll employ to test this architecture. SOA, simply put, is best defined thus:

SOA is a strategic framework of technology that allows all interested systems, inside and outside of an organization, to expose and access well-defined services, and information bound to those services, that may be further abstracted to orchestration layers and composite applications for solution development.

The primary benefits of a SOA, and so the objective of a test plan, include:

- ***Reuse*** of services, or the ability to leverage application behavior from application to application without a significant amount of re-coding or integration.
- **Agility**, or the ability to change business processes on top of existing services and information flows, quickly, and as needed to support a changing business.
- **Monitoring,** or the ability to monitor points of information and points of service, in real-time, to determine the well being of an enterprise or trading community. Moreover, the ability to change processes or adjust processes for the benefit of the organization in real-time.
- **Extend Reach,** or the ability to expose certain enterprise processes to other external entities for the purpose of inter-enterprise collaboration or shared processes.

What is unique about a SOA is that it's as much a strategy as a set of technologies, and it's really more of a journey than a destination. Moreover, it's a notion that is dependent on specific technologies or standards, such as Web Services, but really requires many different types of technologies and standards for a complete SOA. All of these must be tested.

Figure 1 represents a model of the SOA components, and how they're interrelated. What's key here is that those creating the test plan have both a macro understanding of how all the components work together as well as how each components exists unto itself and the best approach to testing those components.

You can group the testing domains for SOA into these major categories:

- Service-Level Testing
- Security-Level Testing
- Orchestration-Level Testing
- Governance-Level Testing
- Integration-Level Testing

I'm going to focus more on service-level testing, since it's critical to SOA. Moreover, the categories or domains that you choose to test in your architecture may differ due to the specific requirements of your project. Moreover, there are other areas that need attention as well, including quality assurance for the code, performance testing, and auditing.

Service-Level Testing

In the world of SOA, services are the building blocks, and are found at the lowest level of the stack. Services become the base of a SOA, and while some are abstract existing "legacy services," others are new and built for specific purposes. Moving up the stack, we then find composite services, or services made up of other services, and all services abstract up into the business process or orchestration layer, which provides the agile nature of a SOA since you can create and change solutions using a configuration metaphor. It's also noteworthy that, while most of the services tested in SOAs will be Web Service-based, it's still acceptable to build SOAs using services that leverage other enabling technologies such as CORBA, J2EE, and even proprietary approaches.

When testing services, you need to keep the following in mind: ***Services are not complete applications or systems and must be tested as such***. They are a small part of an application. Nor are they subsystems; they are small parts of subsystems as well. So you need to test them with a high degree of independence, meaning that the services are both able to function properly by themselves, as well as part of a cohesive system. Indeed, services are more analogous to traditional application functions in terms of design, and how they are leveraged to form solutions, fine- or course-grained.

The best approach to testing services is to list the use cases for those services. At that point you can design testing approaches for that service including testing harnesses, or the use of SOA testing tools (discussed later). You also have to consider any services that the service may employ, and so be tested holistically as a single logical service. In some cases you may be testing a service that calls a service, that calls a service where some of the services are developed and managed in house, and some of them exist on remote systems that you don't control. All use cases and configurations must be considered.

Services should be tested with a high degree of autonomy. They should execute without dependencies, if at all possible, and be tested as independent units of code using a single design pattern that fits in other systems that use many design patterns. While all services can't be all things to all containers, it's important to spend time understanding their foreseeable use and make sure those are built into the test cases.

Services should have the appropriate granularity. Don't focus on too fine-grained or too loose-grained. Focus on that correct granularity for the purpose and use of the SOA. Here the issues related to testing are more along the lines of performance than anything else. Too finely grained services have a tendency to bog down due to the communications overhead required when dealing with so many services. Too loosely grained and they don't provide the proper autonomic values to support their reuse. You have to work with the service designer on this one.

So, what do you test for in services? First, it's important to follow a few basic principles.

First and foremost, services should be tested for reuse (reusability). Services become a part of any number of other applications and must be tested so they properly provide behavior and information but not be application- or technology-specific. This is a difficult paradigm for many developers since one-off custom software that digs deeply into native features is what they've been doing for most

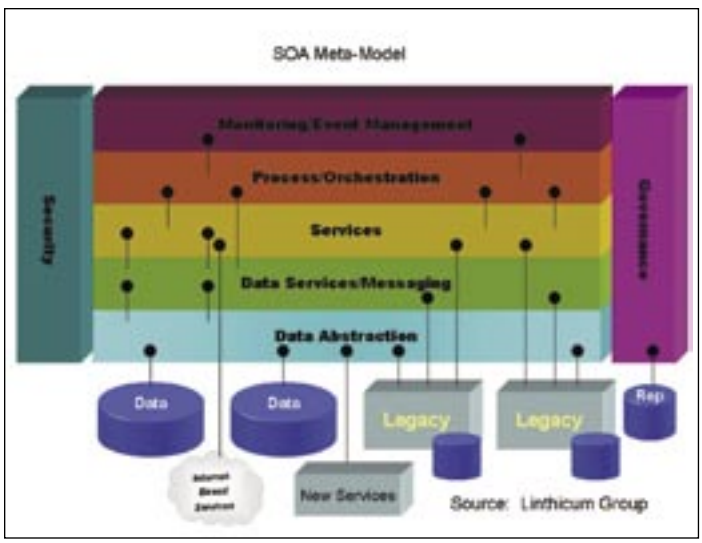


Figure 1: The SOA meta model defines the components of a SOA and how they're interrelated. It's important to understand this before you create a test plan and begin testing.

of their careers. So the patterns must be applicable to more than a single problem domain, or application, or standard, meaning you must have use for your reusable service and it must be in good working order.

To test for reusability you must create a list of candidate uses for the service. For instance, a shipping service that plugs into accounting, inventory, and sales systems (see Figure 2). Then the service should be consumed by the client, either through a real application (in a testing domain) or a simulator, and the results noted.

In addition, the service should be tested for *heterogeneity*. Web Services should be built and tested so that there are no calls to native interfaces or platforms. This is due to the fact that a service, say one built on Linux, may be leveraged by applications on Windows, Macs, and even mainframes. Those that leverage your service should do so without regard for how it was created, and should be completely platform-independent. The approach to testing this is rather obvious; simply consume the service on several different platforms and note any calls to the native subsystems.

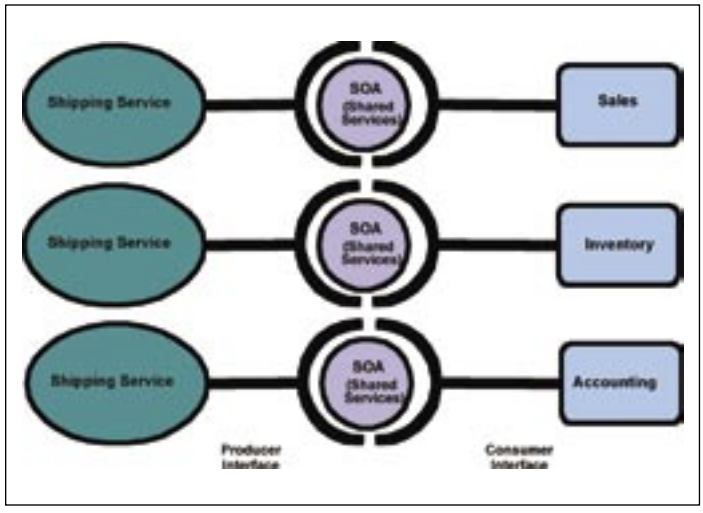


Figure 2: When testing services (producer), make sure to test the service within each application/service (consumer) where the service will have value in the SOA

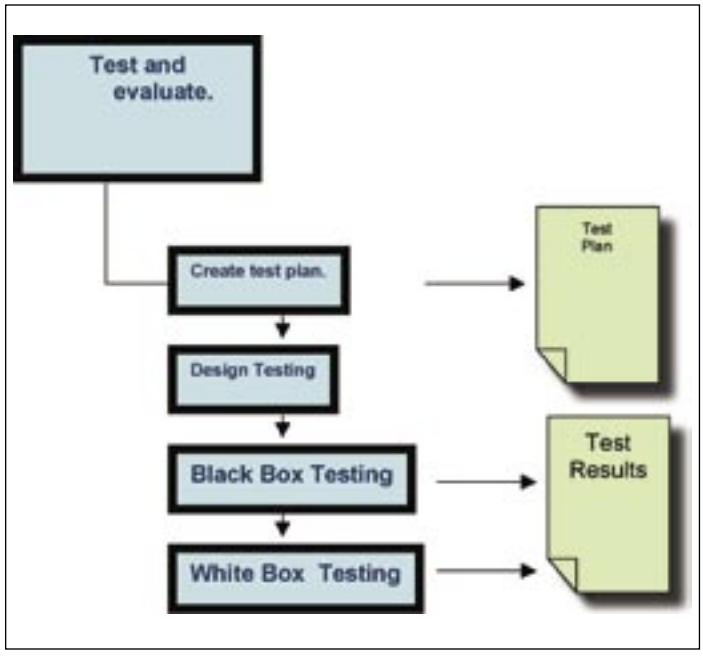


Figure 3: A typically high-level SOA testing process.

You should also test for *abstraction*. Abstraction allows access to services from multiple simultaneous consumers; hiding technology details from the service developer. The use of abstraction is required to get around the many protocols, data access layers, and even security mechanisms that may be in place, thus hiding these very different technologies behind a layer that can emulate a single layer of abstraction.

Abstraction is tested effectively by doing, meaning implementing instances and then testing the results. Regression and integration testing is the best approach, from the highest to the lowest layers of abstraction.

When we build or design services we need to test for *aggregation*. Many services will become parts of other services, and thus composite services leveraged by an application, and you must consider that in their design. For instance, a customer validation service may be part of a customer processing service, which is part of the inventory control systems. Aggregations are clusters of services bound together to create a solution, and should be tested holistically through integration testing procedures.

Service testing means different things to different organizations due to the fact that SOA is so new. Most people who are testing services attempt to figure it out as they go along, typically rolling their own tools for testing, including service consumption test harnesses and service producer test harnesses for the particular use cases they are testing. Others are learning to leverage off-the-shelf testing tools, such as Parasoft SOAtest or Mindreef SOAPscope.

Security-Level Testing

Security strategy, technology, and implementation should be systemic to a SOA and even bring along new concepts such as identity management. When testing your SOA for security issues, the best approach is first to understand the security requirements, and then design a test plan around those requirements, pointing at specific vulnerabilities. Most are finding that black box testing is the best way to test for security issues in the world of SOA, including penetration testing, vulnerability testing, etc., using existing techniques and tools.

A further concern of SOA security is the fact that many SOAs allow services to be consumed outside the enterprise and so create a new set of vulnerabilities including information security issues and denial of service attacks. Moreover, many SOAs also make the reverse trip, allowing for the consumption of services outside of the firewall into the SOA. That opens the door for other types of attacks and the security needs to be tested in this case as well. Vulnerabilities here include malicious services.

Again, most people who are testing their SOA for security issues have a tendency to roll their own approaches and build there own tools. However, a few new tools are appearing on the market such as Vordel SOAPbox to test the security of XML applications, such as XML Web Services. It's used during development and deployment phases to test an XML application's compliance to security standards. SOAPbox highlights security tokens, signatures, and encrypted content in XML documents.

Sound Architecture Requires Proper Planning

WEB AGE SOLUTIONS - YOUR TRAINING PARTNER FOR SOA

In all phases of SOA migration, Web Age Solutions provides training and customized services from awareness to implementation. We support vendor specific or generic SOA training tailored to your organization's needs.

Custom training for complementary SOA technologies

XML • WEB SERVICES • WSDL • SOAP • WEBSPHERE • WEBLOGIC • JBOSS • J2EE • SPRING/HIBERNATE/STRUTS • WID/WBI/WMB

Custom training plans for virtually every job role

BUSINESS ANALYST • ADMINISTRATOR • DEVELOPER • ARCHITECT • QA/TESTER • MANAGER • EXECUTIVE • SECURITY

Consulting services for all phases of SOA migration

IBM Premier Business Partner

Add Web Age Solutions to your plan & stay ahead of the competition

www.webagesolutions.com - 877.517.6540 - soa@webagesolutions.com

WEB AGE solutions inc.

Orchestration-Level Testing

For our purposes, we can define orchestration as a standards-based mechanism that defines how Web Services work together, including business logic, sequencing, exception handling, and process decomposition, including service and process reuse.

Orchestrations may span a few internal systems, systems between organizations, or both. Moreover, orchestrations are long-running, multi-step transactions, almost always controlled by one business party and are loosely coupled and asynchronous in nature.

We can consider orchestration as really another complete layer over and above more traditional application integration approaches, including information- and service-oriented integration. Orchestration encapsulates these integration points, binding them together to form higher-level processes and composite services. Indeed, orchestrations themselves should become services. So you test them as you would other services, including abstraction, reuse, granularity, etc. However, you should note that these services sit above existing services and the testing should regress from the top-level services down to the bottom-level services. Or from the orchestration layer down to the primitive services. Tools such as Mindreef SOAPscope: Solutions for Web Services and SOA may be effective here as well.

Governance-Level Testing

SOA Governance is an emerging discipline, which enables organizations to provide guidance and control of their SOA initiatives and programs.

Although there are many SOA governance vendors and thus many SOA governance definitions, the best is from Wikipedia.

Many organizations are attempting to transition from silo-oriented applications to agile, composite clients and services. This transition requires that the 'service' become the new unit of work. The IT organization must now manage these services across the entire lifecycle, from inception through analysis, design, construction, testing, deployment, and production execution. At each stage, certain rules or policies must be carried out to ensure that the services provide value to the consumers. SOA governance is the discipline of creating policies, communicating, and enforcing them.

So, in essence you have a lifecycle and policy management layers, including testing, that needs testing. No problem. You test governance systems by matching up the policies the governance system is looking to manage and control with the actual way they manage and control them. Thus, it's a simple matter of listing the policies and establishing test cases for each. Such as:

XYZ Service can only leverage services within the firewall test policy

Does the policy disallow that service from leveraging services outside the firewall?

Easy as that, and there are no tools for testing SOA governance systems other than those provided by the SOA governance solution vendors.

Integration-Level Testing

As with traditional integration testing, the purpose of this step is to figure out if all of the interfaces, including behavior and information sharing between the services, are working correctly. The type of integration testing you carry out should work through the layers of communications. Working up through the network to the protocols and inter-process communications, including testing the REST or SOAP interfaces to the services or whatever communication mechanism are employed by the services you're deploying.

Things to look for here include:

- Can communications be established with late binding, meaning dynamically as-needed?
- Is the integration stable under an increasing load?
- Is the transmitted information correct for the service or applications?
- Are the security mechanisms working properly?
- How does the SOA recover from application, database, and network failures?

Creating a Test Plan

The test plan you create for your SOA should reflect the requirements of your project, and, unfortunately, one size does not fit all. Figure 3 depicts the high-level process you can employ to drive SOA testing for your project. However, you may have special needs such as more emphasis on performance and security.

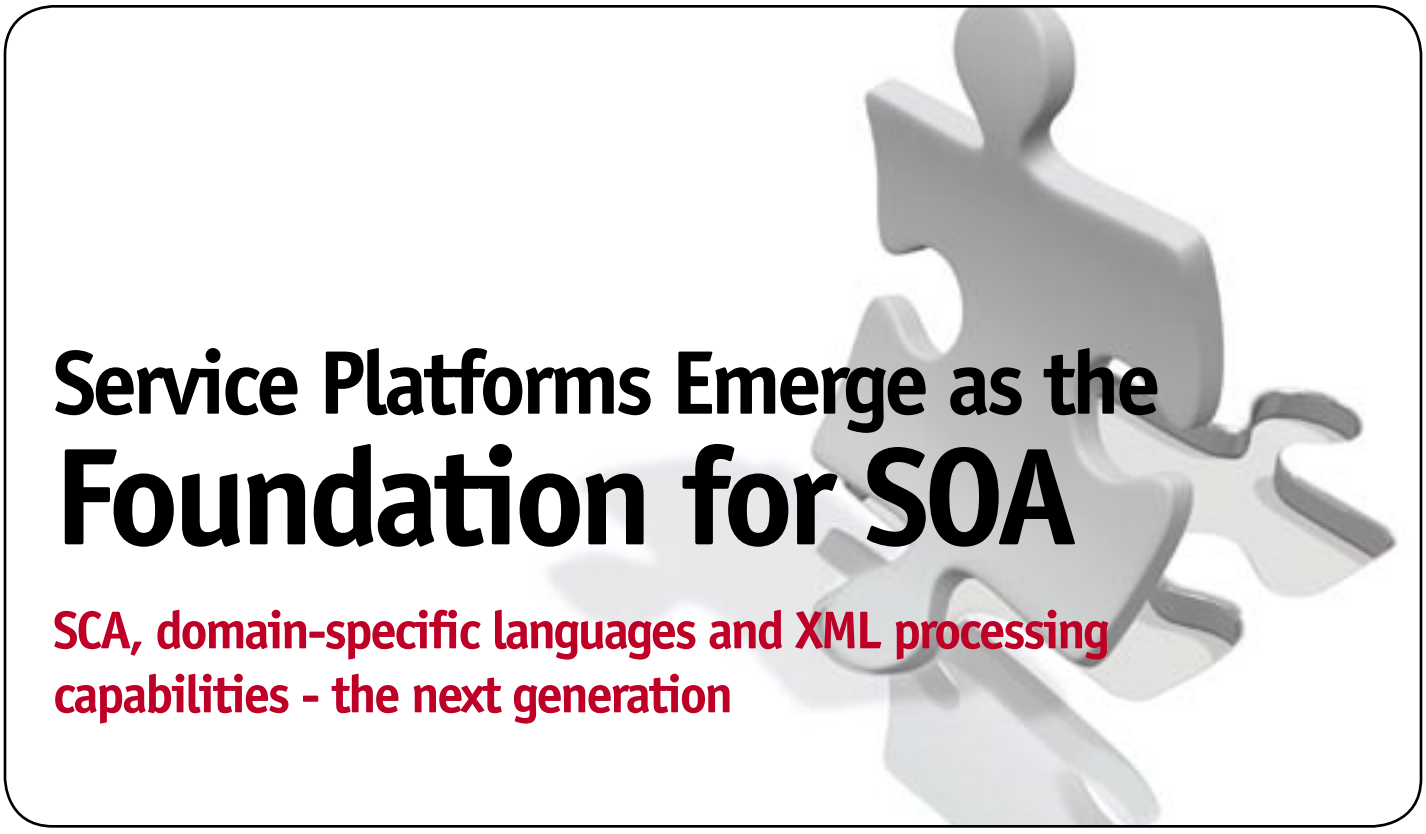
At the end of the day you'll find that SOA testing incorporates all of the testing technology and approaches we've developed over the years for other distributed systems and adds some new dimensions such as services, orchestration, and governance testing. Those who understand testing will have to expand their skills and understanding to include those new dimensions. Many failed SOA projects are directly attributable to lack of testing...many assuming that the new technology would work flawlessly. Unfortunately, that's just not the real world yet. ■

> **"Security strategy, technology, and implementation should be systemic to a SOA and even bring along new concepts such as identity management"**

About the Author

David S. Linthicum is the CEO of the Linthicum Group, LLC, a SOA consulting and advisory firm focusing on the real-world design, development, implementation, and testing of Service Oriented Architectures

david@linthicumgroup.com.



Service Platforms Emerge as the Foundation for SOA

SCA, domain-specific languages and XML processing capabilities - the next generation

WRITTEN BY **DEMED L'HER AND GREG PAVLIK**

Enterprise software architectures are shifting from collections of applications that are designed around user interfaces to assemblies of reuseable services. The first step in the evolution toward service-based applications was the definition and publication of services encapsulating discrete business functions.

The second wave used services in point-to-point combinations using protocols aimed at system interoperability for communication. The next wave of SOA adoption will focus on enabling composite service definitions that combine domain-specific languages for process orchestration, XML transformations, message routing, and business rules.

In this article, we'll look at how SOA platforms are evolving to meet these requirements. Specifically, we'll examine three related themes:

1. The nature and role of service platforms that are designed to host composite services and complex business processes;
2. The changes in how applications are described and designed in the new SOA platforms;
3. The importance of key standards in simplifying and commoditizing the integration of services and applications.

In particular, we'll look at a series of emerging industry standards that describe how to design composite services implemented using many different implementation languages and protocols. These standards are defined in the Service Component Architecture (SCA) framework.

Why Services?

Interest in architectures based on services is driven by three distinct yet complementary technical goals. These goals include the need to:

1. *Integrate software functions across the data center to provide a consistent and rationalized approach to dealing with enterprise integration scenarios.* Data and business functionality is often bound up in silos that need to be bridged to create new business functions that use multiple back-end systems. While initial approaches to Enterprise Applications Integration (EAI) often included ad hoc designs and layers of proprietary infrastructure, this is changing: a model based on the service paradigm and open standards is becoming the norm.
2. *Expose software assets as stateless services based on open Internet standards.* Traditional systems integration required a uniform substrate to connect endpoints. We have reached a point in the industry where virtually any software asset can be exposed as a SOAP-based Web Service and described in WSDL.
3. *Leverage discrete services to build new business processes bridging many systems together.* The standard language for building business processes is BPEL. However, effectively building a composite

service out of multiple existing systems often requires additional functionality, including business rules, declarative XML processing capabilities, and asynchronous business events. The key requirement for service platforms is to support the composition of multiple services using multiple technologies required to implement composite services.

These technical goals are mirrored by specific business benefits that can be traced directly to SOA adoption. The business benefits start with increased flexibility to leverage and maximize the value of an organization's IT assets. SOA also increases productivity by using standards and reducing the effort that is required to get services communicating with each other.

Lastly, the SOA approach is comprehensive: it's a model that can tie together many technologies and describe ways in which services are related to each other. All these factors lead to cost savings.

Example

Figure 1 shows a fictional HR system architected in a Service Oriented Architecture fashion. This system provides an end-to-end solution for employee provisioning, including a front-end to capture employee details, BPEL processes to orchestrate the provisioning of the employee assets, and a human workflow to route and gather manager approvals through e-mail.

An HR representative enters the new employee details on the HR Web site. This results in the publication of a message containing these details on the enterprise service bus. The ESB then looks at the country of hire of this new employee and routes it to the appropriate BPEL process to take into account the fact that HR regulations and practical details such as office space greatly differ from one location to another.

The "Employee Creation" BPEL process involves various other systems. First, it creates a new entry in the HR database. Then, it invokes an external rules engine to find the level of approvals required for a new employee based on such criteria as grade and department.

The next step is to gather the appropriate approvals, and this task is orchestrated with a human workflow that will take care of e-mailing managers. Finally, once all approvals have been received, the BPEL process publishes an event on the bus to trigger various other provisioning processes: IT provides the employee with all required internal accounts such as e-mail, payroll sets up the employee in Oracle Financials, and facilities allocates office space.

The benefits of designing this employee provisioning in this greatly decoupled fashion are:

- Because there is a single core employee creation BPEL process per country, the task of designing this process is greatly simplified and takes into account vastly different local regulations. A single application or process that tries to overlay all these variations would result in extremely complex logic.
- Because there is one process per country, adding a new location is a much less risky task: there's no need to touch the running processes for the existing locations.
- Because the approval levels are externalized to the rules engine, HR representatives can easily update the rules as hiring guidelines evolve, without having to modify the core BPEL employee creation process – a task that would require developers and testers to be involved.

But these benefits come at the price of having to manage many different types of artifacts: BPEL processes, rules definitions, and ESB flows to name a few. In many cases, this means running many different kinds of middleware that are packaged, deployed, and administered separately. The cost of the flexibility in the design comes in the form of much greater management, administration, and governance. The new SOA platforms and standards are designed to eliminate these costs by providing one model for building and running composite services.

In the following sections we see how SCA provides a model that lets us combine these different technologies together into a single composite service definition – a key requirement for SOA solutions.

Service Component Architecture

Until recently, the development of composite services was hampered by a reliance on proprietary models for building new services and business processes. The development and acceptance of BPEL as a standard for service orchestration was a major step forward for the industry. The Service Component Architecture (SCA) is the next step in that evolution. In fact, we believe that the SCA will be viewed in retrospect as the key enabling technology for the widespread, successful adoption of SOA.

The SCA is a family of specifications developed by a group of leading vendors and platform providers in the integration and applications spaces. In February 2007 an initial set of SCA 1. 0 specifications completed incubation and were published on www.osoa.org. The authors announced their intention to submit the specifications to OASIS' open standards process. In addition a new member section, the Open Composite Services Architecture (OpenCSA) Member Section http://www.oasis-opencsa.org, was created in OASIS to coordinate the several technical committees that will start work later this year to process these specifications.

These SCA specifications support the construction of composite

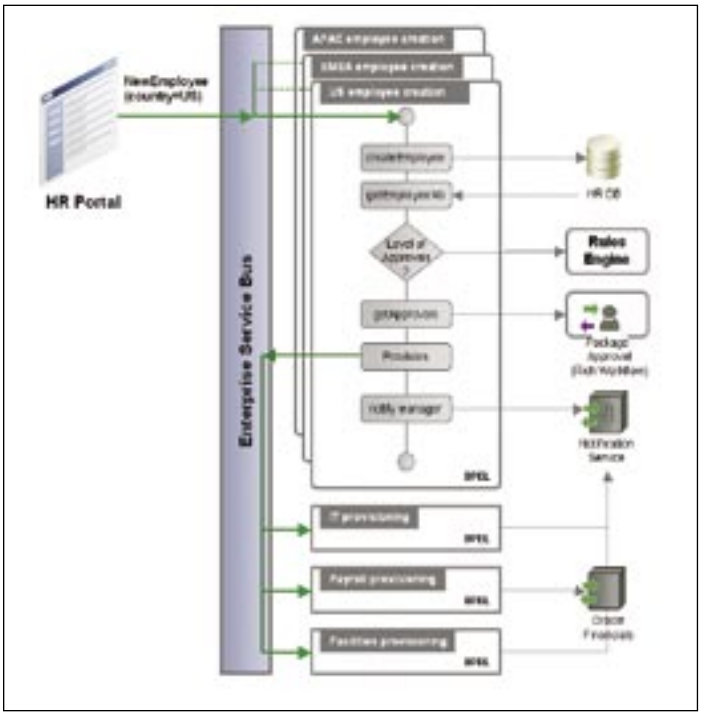


Figure 1: A fictional HR system architected in a Service Oriented Architecture fashion

services using many different technologies, including BPEL, XML, Java, C/C++, PHP, as well as offering extensions for platform-specific technologies. The glue that holds all of this together is the SCA Assembly specification. The SCA Assembly model acts as a kind of "global deployment descriptor" that describes all of the artifacts that are used to build a composite service and how they relate to each other.

The key components of the assembly model are:

1. Composite

A composite is the encapsulating abstraction for the description of a composite service and is described in XML. The rest of the SCA assembly artifacts that we describe are contained in a composite.

2. Service

The service defines the "entry point" for a composite service. The service consists of an interface that is generally defined as the PortType of a WSDL document. It also includes a binding element that describes the protocols that are used to contact the service. For example, SOAP 1.1 is a common binding used to make composite services network-accessible. Bindings can also contain policies that are enforceable rules that further refine the requirements and capabilities of a service.

3. Component

A component is a piece of encapsulated business logic that provides some or all of the functionality offered by the service. A component is an abstract, technology-independent description of some concrete functionality that is hosted in the service platform runtime. Components are configured to point to implementations of business logic, which can be implemented with different languages or technologies. For example, a component may be used to configure and load a BPEL process into the service platform. Components implement an interface that can be matched up to a service element to make the component accessible. Typically, a component has one or more references that are connected to other components or external services.

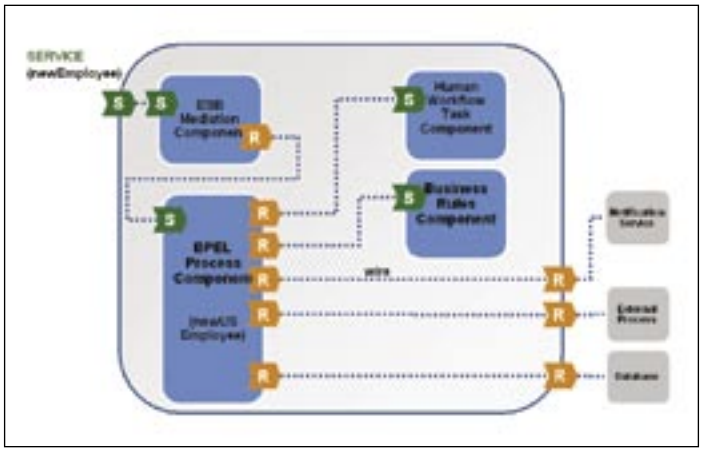


Figure 2: A Composite service defined using the Service Component Architecture

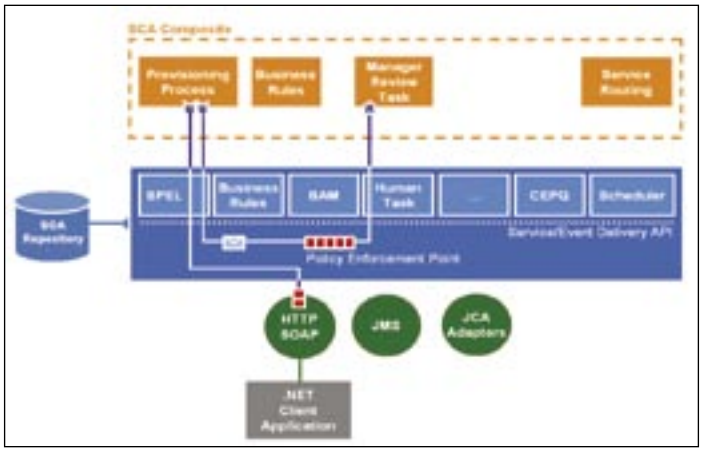


Figure 3: A SOA platform supporting the Service Component Architecture

4. Implementation

The implementation is the actual business logic or declarative XML processing rule that is executed by the platform when a message is dispatched to the component. An example of an implementation is a BPEL process definition or a Java class file.

5. Reference

A reference refers to services that are outside of a composite. Like a service, it supports a particular interface, protocol binding, and policies.

6. Wire

Everything described so far is connected together in the SCA Assembly model by wires. A wire is a connection that links together services, components, and references. Wires don't have special semantics: they are used to tell the service platform how to route messages between the network layer and the pieces of business logic that are used to implement the composite service.

The employee provisioning service (discussed above) can be modeled as a composite in SCA. Figure 2 shows a logical representation of the composite. In this case, the composite's service is wired to a component representing the core BPEL process. The partner links in the BPEL process are resolved by wires to human tasks, business rules, and external services. Instead of a combination of different middleware platforms, this example shows a single composite that combines together many building block technologies to build one service.

Finally, while other standards such as WSDL help to define the description of services, SCA also defines how these services interoperate through bindings and policies. In our example, the composite service and reference elements would include bindings that describe the protocols used by the service when it's made available on the network. SOAP is an ideal default binding type to use with SCA, as it maximizes system interoperability and can also easily be optimized under the cover (for instance for intra-JVM communications). Ultimately, this SCA model, coupled with SOAP and WSDL, results in more uniform and interoperable applications.

SOA Runtime Platforms

A composite service designed with SCA needs to be deployed to a SOA platform. It may be helpful to take a look at a platform architecture that can be used to host composite services. Figure 3 shows a representative SOA platform architecture.

Using containers, the platform architecture supports many do main-specific languages for implementing business logic. These languages include process and orchestration capabilities, ESB functions such as service virtualization and message routing, human task management, and business rules. The SCA model provides a

REALWORLD JAVA SEMINAR
REALWORLDJAVA.COM

THE LARGEST JAVA DEVELOPER EVENT ON THE EAST COAST

Register Today!
Early Bird: Save $100
www.realworldjava.com

A Roadmap for Java Professionals

Hands-On Education:
The Largest Java Developer Event on the East Coast

A seasoned Java professional has to know more than just the syntax of the Java language. Java EE offers a set of standardized technologies for enterprise development. A number of open-source frameworks such as Spring or Hibernate are widely used in a variety of Java applications. Familiarity with new "beyond-Java" languages and technologies will widen your horizons and make you a more valuable Java professional.

— Sponsored by —
No. 1 i-Technology Magazine in the World
JDJ

— Produced by —
SYS-CON EVENTS

For this and other great events visit www.EVENTS.SYS-CON.com

LEARN FROM THE MASTERS:

- Java 6.0 - New Features
- EJB 3, Spring and Hibernate: A Comparative Analysis and Recommendations
- Adobe Flex for Java Developers
- Enterprise Service Bus as a Centerpiece of SOA Implemented with Java
- Code Quality: Pay Now or Pay Later
- EJB 3. and Java Persistence API
- Concurrent Programming in Java
- XML Processing in Java
- Programming with Spring Framework
- AJAX for Java Developers
- Ruby on Rails for Java Developers

standard way to combine these capabilities together and deploy them to the platform. The containers manage the implementation artifacts and execution for each implementation type.

Similarly, SOA platforms will support many protocols for connecting to back-end services and integrating with pre-existing systems. The bindings defined in SCA composites seed the protocol-binding layer to enable interoperability across the data center. The main function of the platform itself is to manage the interactions between all the building blocks that are used to host services. This includes deployment and lifecycle functions, system management, routing of messages between the building blocks based on the wiring in the composite, governance, and metadata management. We expect many platforms that leverage SCA for building composite services to emerge on the market over the next year or so. These platforms will focus on declarative processing of XML messages and domain-specific languages for functions like process orchestration.

Summing It Up

The next generation of SOA applications will combine domain-specific languages and XML processing capabilities, from Java to BPEL, leveraging the wide array of tools and standards available for developing services. The enabling technology for assembling these distinct technologies into composite services is the Service Component Architecture. As Service Oriented Architectures become more ubiquitous in modern data centers, SCA-based SOA platforms will be the de facto standard for building and hosting services. These trends will shape the future of application development, much like J2EE standards helped define enterprise Web development in the 1990s ■

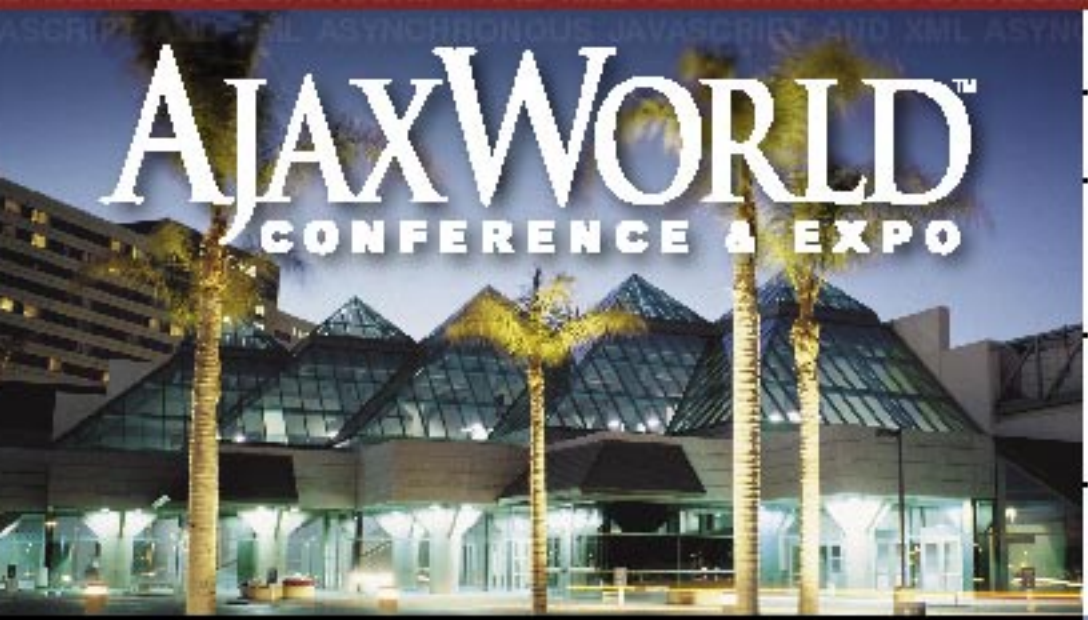
About the Authors

Demed L'Her is a senior principal product manager at Oracle. His focus is on Enterprise Service Buses, JMS, and next-generation SOA platforms. He has been involved in messaging and integration projects worldwide for 10 years.

Greg Pavlik is an architect at Oracle. In this role he works on a combination of technology strategy, product development, and standards. He is currently responsible for Oracle's SOA and Web Services offerings. Greg is also the author of Java Transaction Processing (Prentice Hall, 2004).

Web 2.0 and Rich Internet Apps

CALL FOR PAPERS NOW OPEN!



Providing developers and IT managers alike with comprehensive information and insight into the biggest paradigm shift in website design, development, and deployment since the invention of the World Wide Web itself a decade ago.

On September 24-26, 2007 over 1,000 developers, architects, IT managers, and software professionals of every stripe will be converging in Santa Clara, CA to attend the West coast AJAXWorld Conference & Expo -- the most comprehensive meeting on the most significant technology subjects of recent times: AJAX, Rich Internet Apps & Web 2.0.

Experience AJAX, RIA, and Web 2.0 Knowledge and Best Practices at AJAXWorld Conference and Expo 2007

Delegates will hear first-hand from the creators, innovators, leaders and early adopters of AJAX, and a slew of leading vendors will provide sneak-peeks of the very latest frameworks, tools, products and applications during the conference, which will have over 100 sessions and presentations given by 125 different speakers - the largest AJAX-focused speaker faculty ever assembled in one place at one time.

AJAXWorld Conference & Expo 2007 West will be jump-started with a full one-day "AJAX University Boot Camp," followed by the Main Conference and Expo over the following 2 days.

Highlights Include:

- *100+ conference speakers selected from among the world's leading AJAX technologists and practitioners, including high-level IT executives, industry thought leaders, VCs and analysts*
- *Topical, Interactive "Power Panels" simulcast on SYS-CON.TV*
- *Dozens from 30+ vendors both in General Session and in the Exhibit Hall/AJAX Showcase*
- *Pre-Conference "AJAXWorld University Bootcamp" for those wanting an all-day, hands-on encounter with AJAX*
- *Leading-edge sessions on issues like Offline AJAX, Mobile AJAX, Enterprise AJAX, AJAX Security, Desktop AJAX, Semantic Mash-Ups, Next-Generation SaaS, JavaScript & XML, Google Web Toolkit, Adobe Apollo, Adobe Flex, AJAX RIA GUIs, and Appropriate vs Inappropriate Use of AJAX.*

Join Over 3,600 Early AJAX Adopters Who Attended the #1 Web 2.0 Event in the World!

Early Bird Price: $1,495
(Register Before May 25, 2007)

SYS-CON EVENTS
For more great events visit www.EVENTS.SYS-CON.com

VISIT WWW.AJAXWORLD.COM FOR THE MOST COMPLETE UP-TO-DATE INFORMATION

SOA and Service Virtualization

Building the common plumbing required by all services-based applications

WRITTEN BY **ROURKE MCNAMARA**

Service Oriented Architecture (SOA) promises to reduce the amount of new code required to create new applications by allowing the reuse of existing services. To get significant benefit from SOA, an organization must have as many services exposed as possible at as broad a level as possible.

Those services will be very expensive to manage if they are all written from the ground up and don't build on a common framework for communication, deployment, and management. Civil engineers don't build an office tower with independent plumbing, electricity, and insulation for each office. Services shouldn't be built that way either.

Development organizations need to respond more quickly to changing market requirements in today's ultra-fast-paced business world. SOA promises to help those IT departments drastically shorten the development time for new initiatives while increasing the stability of the delivered functionality. How can a new approach to building applications do that?

To reuse services, companies must first service-enable existing assets and build services to meet the needs of ongoing business initiatives. With each project built according to SOA principles, the library of services available to the next project will grow. As that library grows, so will the benefits of SOA.

Unfortunately, as that collection of services grows, so does the complexity of the environment in which they're deployed. Services-based applications are, by their nature, more complex than standalone applications. At the very least, a services-based application consists of a number of different components (services) that all need to communicate for that application to function right. This communication has performance implications, introduces new failure points and can weaken security.

Services-based applications developed using the same techniques as standalone applications will also have significantly more code than monolithic applications with equivalent functionality. This is because conventional service development tools require that communication between services be hand-coded in each and every service that's built. Inter-service communication is fundamental to service-based architectures and shouldn't have to be hand-coded each time a new service is built.

Application servers provide a framework for building Web-based applications that lets developers focus on building the flow and logic of a Web application. Functionality that's common to all Web applications – such as resource pooling, thread management, and session management – is not hand-coded by developers for each application. Instead, the application server provides for this common functionality. In SOA, communication between services is common across all servers and should not be hand-coded, but should be provided by the deployment environment for those services.

Service Virtualization Defined

Recent studies show that 40% of the code written for the average service developed using an application server is unnecessary "plumbing" code.

Large office buildings don't have duplicate plumbing systems and the business units of a large corporation that inhabit those buildings don't build duplicate HR departments. In SOA, the same principle applies. Services must be reused across as much of the organization as possible to get the full benefit of SOA.

As a company gets to be 10 or 15 years old it doesn't create a second human resources department because the original one is "out-of-date." The same principle applies to old services. Modernization might be required, but replacing services should be avoided whenever possible. This lets developers focus on delivering new value to the business organization.

Services with very broad reuse and services with long life spans both imply a heterogeneous service environment. No company can standardize on a single development language or toolkit and expect that they will use it forever. Internal personalities, corporate mergers, and technological progress will all contribute to a library of services based on different programming languages and frameworks.

Heterogeneous environments make sense. The computers found that in the typical corporate office there are a mix of different form factors (laptops, desktops, servers) that run on different operating systems and run different software packages. However, those computers all connect to the same network and power infrastructure.

The same must be true of services. A services-based application can certainly be an orchestration leveraging a Java service, a .NET service, and a C++ service. All of those services should communicate, be deployed and monitored using the same mechanisms.

Service virtualization is an approach to deploying and managing services that provides the common plumbing required by all services-based applications. This lets developers focus on building new functionality and provides a foundation on which you can plug that new functionality.

To do this, new services must be written to run as components in a service container. That container must then provide a straightforward mechanism by which the newly deployed service can both be invoked, and invoke other services. This clear separation between the service logic and its communication with other services is the most important requirement for effective service virtualization.

ESBs and Beyond

Enterprise Service Bus (ESB) products have already been implemented by many companies with SOA initiatives. These products provide enormous value by allowing companies to easily bridge communications between services, packaged applications, and legacy assets. Complete ESB offerings also provide a mechanism by which legacy systems can be intelligently exposed as services without any changes to those existing system. Many even go so far as to providing the ability to very quickly orchestrate a set of services into a higher-level business service using a graphical flow language.

These ESB products all provide for mediation as their core functionality. This mediation allows ESB customers to remove the in-code coupling between services. Some companies are currently plugging all of their services directly into these products. By combining the out-of-the-box connectivity features found in ESB products with messaging technology, standards, and conventions these companies have implemented the most basic form of service virtualization.

The ESB-based service virtualization approach helps solve much of the communications problem. However, the management of those services isn't addressed because ESBs only address communication between services and don't provide a container in which to host the service. Further, developers still need to write some communications code because the ESB only mediates communication.

Implementing Service Virtualization

Developers should be able to focus simply on writing the functionality required for a new business initiative or business application. Once that service has been deployed, a service virtualization container should:

- Provide the communication required to invoke other services;
- Allow the service to be invoked from orchestrations and other services;
- Insure data integrity across multiple service invocations;
- Secure all communications according to corporate requirements;
- Control access to the service;
- Track use and performance of the service;
- Provide common pooled access to resources (for example, DBs, JMS);
- Provide for hands-off migration of services from dev-test-production; and
- Allow for lifecycle operations.

A service container that can only host Java code isn't enough. Managing large-scale service networks requires that there be a single notion of service and a single mechanism for deploying, monitoring, and managing all services regardless of programming language or toolkit. This means that Java, .NET, and ESB services should all be running in the same kind of container.

Companies in such heterogeneous environments struggle with the very definition of "service." Is a service anything with a clearly defined interface? Is it something with a SOAP interface and a WSDL definition? Is an orchestration a service? Service component architecture (SCA) finally provides a single definitive answer to these questions by providing a universal definition of "service." It provides not only a formal interface definition, but also ties that interface to an implementation, specifies service and non-service dependencies, and allows deployment overrides to be written in a standardized way. Also unlike WSDL, SCA doesn't tie the service to a particular communications endpoint. This is very important because it further decouples service code from the underlying plumbing.

It's not enough to provide a single definition for a "service" and a single model for describing that service. To have a single container that can host heterogeneous services there has to be a standard by which all of those services can "plug-in" to a container.

Just as the Enterprise Java Bean (EJB) specification provided a standard for plugging code into an application server, Java Business Integration (JBI or JSR208) provides the standard by which services can easily plug into a service container or service virtualization environment. JBI describes how the service communicates with the service container and also how the service is monitored and managed.

A service container built on a combination of SCA and JBI lets developers write services with no transport code at all. Using other services becomes as easy as writing a single line of code: create a service object and invoke a method on that object.

—continued on page 12

Closing the Loop

The role of governance in unlocking SOA business value

WRITTEN BY **HUGH TAYLOR**

Governance is the tail that wags the SOA dog. An organization that deploys Web Services without a solid governance program is headed for serious trouble in terms of reliability, security, and cost.

While governance is a necessity for any IT endeavor, the open and potentially chaotic nature of SOA makes governance either a showstopper or a magic bullet for success, depending on how it's approached. This article will look at SOA governance from both a technological and business perspective, highlighting the need for a complete governance model – a closed loop that spans design time to runtime, policy definition, enforcement, and audit.

The Goal: Business Optimization through SOA

Encounters with today's endless discussions about Service Oriented Architecture bring to mind the old Hollywood phrase, "Cut to the chase." Forget the endless ruminations on containers, service buses, standards, and IDEs. What are we really talking about when we get granular with Web Services and their attendant minutia? We're talking about money.

SOA is about money, or at least, it should be. An SOA is a long-term technology initiative (it never really ends) that is meant to deliver a return on investment, whether you're investing hard dollars or equivalent resources. A non-profit with volunteer developers is just as much an investor in an SOA as General Motors.

In particular, SOA should be able to deliver on several of its well-hyped promises for business value, including a reduction in overall IT cost, business agility, and a faster application development and integration cycle. At the root of all these potential savings is the concept of software reuse. If a Web Service can be developed once, but reused many times across multiple consuming applications, then many cost reductions fall into place. The need to redevelop a comparable application drops off, and the cycle time for getting the new application up and running will be shorter if there's no development cycle to go through. Reuse also serves agility by enabling rapid extensions of systems into new uses based on dynamic shifts in strategic and operational requirements.

Getting There: A Path of SOA Governance

Reuse, the main driver of financial value from Web Services, is based on the highly flexible nature of Web Services. However, this flexibility is a double-edged phenomenon. The same openness and flexibility can also make Web Services chaotic and unreliable. How can you reach the financial results of reuse with such a potentially out-of-control mechanism?

Though it may seem counterintuitive, the path to the business benefits of SOA, to all that groovy openness and flexibility, leads right through a world of tight governance. The necessity of adopting strict but agile controls as a prerequisite for freedom is reminiscent of the classic Harry Truman line, "If you want to live like a Republican, you better vote for the Democrats." Only when you have a high level of confidence that your Web Services are well-managed, secure, and governed can you be confident that sharing them broadly won't cause more problems than the freedom is worth.

So, let's think old school IT governance and learn to govern our Web Services. SOA governance is a matter of getting your Web Services and overall SOA to behave in the way it's meant to, and preventing it from being misused. It's about control. If you design a Web Service for a specific purpose and set of consumers, you want to have an assurance (and the ability to audit) that it was only used for that purpose and consumer. And, you want it to be available, performing as intended, and secure. The transactions must have integrity, at least to the degree that you specify.

If you can't control your Web Service to this extent, you haven't governed it, and it will inevitably be misused, become unreliable and insecure. The risks of an ungoverned Web Service, and by extension an ungoverned SOA, include performance failures, security breaches, and breakdowns in data integrity. A low-performing, unreliable, and insecure SOA doesn't generate reuse or agility. It's just a costly pain in the neck. Let's not go there.

An SOA governance program delivers you from these risks. A good governance program will include provisions for the security and reliability of Web Services as well as sophisticated methods for controlling the reuse of Web Services. With governance in effect,

you can exploit the reuse and agility potential of your Web Services. You can assign usage rights to Web Services to new consumers, confident that they're authorized and authenticated for use. You can meet security and reliability requirements set out for Web Services by the consumers you work with.

For example, if you develop a Web Service, and wish to extend it for reuse with a new business partner, you will likely want to be able to set, enforce, and monitor a service level for that Web Service. And, you'll probably want to establish, enforce, and monitor security policies that benefit both you and the partner, regarding the Web Service. Depending on your industry, you may need to encrypt or digitally sign the SOAP messages and establish an audit log of Web Service use and transactions for non-repudiation.

There are three main levers of SOA governance: policy definition, policy enforcement, and audit. To govern, you must be able to define the rules you expect to have followed. That's policy definition. And, to govern, you must be able to enforce the policies you define. It does little good to define policies if you can't enforce them. That would be like leaving a store unlocked and hoping that customers will have the decency to put money in the till on the honor system. Believe it or not, many IT governance programs rely to a degree on the honor system for policy enforcement. Your SOA doesn't have to be that way.

Finally, to satisfy various stakeholders that defined governance policies have in fact been enforced, you need a high-integrity audit record of what has gone on. Besides being able to audit the actual Web Service transactions, you also need to be able to show an audit log of the enforcement of the governance policies themselves. For instance, if you want to show a stakeholder that you specified that all SOAP message emanating from a specific Web Service must be encrypted, you have to be able to produce a bulletproof audit log of the establishment and enforcement of that policy.

Audit-worthiness is of interest to auditors of both the "capital A" and "small a" variety. There are auditors of various sorts in your company and its partners who need assurance that SOA governance is in place. There are also auditors (Capital A) who review your company's financial statements for the SEC and your internal controls for Sarbanes-Oxley.

It's only with these tight controls that the Web Service begins to have any reuse value. The challenge, of course, is to impose all this control without locking the whole SOA up so tight that it defeats its entire purpose. If you just wanted lockdown, you would have stayed with your existing systems. The trick, therefore, is to automate your SOA governance to the point that these controls are highly automated and seamlessly baked into the Web Service lifecycle itself. This is where the "loop" comes in.

Closing the Loop

Runtime SOA governance, the ability to define, enforce, and audit policies for Web Services at runtime, should be a benchmark of SOA maturity. Any serious enterprise SOA should be well governed at runtime. Of course, that itself is not always so easy, as Web Service endpoints and the "last mile" reaching from Enterprise Service Buses to the Web Services themselves can be out of the reach of many built-in ESB governance packages. Ditto for multi-vendor ESB to ESB mediation. However, assuming you've licked those challenges and adopted sound runtime SOA governance, you must still contend with the "loop."

The loop is the complete lifecycle for a Web Service from design time to runtime. The loop can either be "broken," meaning that you define policies at design time and hope they get enforced at runtime, or "closed," where definition, enforcement, and audit happen seamlessly in one solution. If you have two separate governance systems, one for design time and one for runtime, you might find yourself in a "define and hope" form of SOA governance, which is unsustainable and unreliable. Throw multiple platforms and enterprises into the mix and things can get downright out of control. "Define and hope," or loosely integrated design time and runtime governance systems, inhibit the realization of SOA business value.

The broken loop model of SOA governance carries with it a serious risk, which is the potential for a drop-off in governance between design time and runtime. Though this may not be a huge issue for a small organization, it can be a major problem in security and compliance terms for a large or complex operation. If the governance policies defined for a Web Service at design time can't be assured at runtime, and there's not an audit log, then the governance will be unsound. By extension, all of the reuse and agility you want from that Web Service will be very difficult to realize. At the very least, it will be quite costly to manually supervise the continuity of governance across the broken loop.

If you opt for a "closed loop" approach to SOA governance, you can define policies at design time and automatically enforce them at runtime. Closed loop SOA governance typically leverages the power of UDDI registry and a policy metadata repository that functions across the full life cycle of the Web Service. With a closed loop governance solution, your enforcement of defined policies will be seamlessly auditable from design time through runtime. With a closed loop approach, realizing the business value of SOA comes within reach. You have the ability to reuse Web Services and perform integrative acts of agility but stay confident that the governance policies you need to assure the integrity of your business remain in place.

Conclusion

SOA governance is a critical activity for an enterprise seeking to find the business return on its investment in Web Services. Though it may seem counterintuitive, the best way to find the financial upside in the openness and flexibility of Web Services is to implement tight governance. To work, though, the governance must not be overly restrictive of change. Otherwise, it would defeat the whole purpose of the SOA.

Best practices suggest that a comprehensive SOA governance solution should encompass both design time and runtime. Ideally, the solution should provide a "closed loop" approach that automates and connects the definition of governance policies at design time with their enforcement and audit at runtime. When the loop is closed, then the agility of SOA can take center stage with a minimum of governance overhead but a high level of confidence in reliability and security. ■

About the Author

Hugh Taylor is vice president of marketing at SOA Software, a provider of management and security solutions for enterprise Service-Oriented Architecture. He is the co-author, along with Eric Pulier, of Understanding Enterprise SOA (Manning, 2005). He has written more than a dozen papers and articles on the subject of Web services and Service-Oriented Architecture. Hugh received his BA from Harvard College and his MBA from Harvard Business School.

*hugh.taylor@soa.com*Immersive SOA

Cretaceous Software:

Collaborative SOA Prototyping with the *Delve SOA Fast-Prototyping Toolkit*

Immersive SOA

Delve is a collaborative fast-prototyping SDK for Tectonic, a service-oriented language. It is intended for use in the early stages of an SOA project, when architects, business analysts, and designers are focused on brainstorming prospective XML data formats and service interfaces.

Use of Delve makes it possible to employ an architectural strategy known as Immersive SOA. Practitioners of this strategy make service-oriented concepts first and foremost when approaching requirements and design tasks. This means, for instance, that a newly proposed service is conceptualized, described, and designed in an "interface-first" manner, with definition of XML-Schema types and a service WSDL preceding any development work. Tectonic's service-oriented nature makes its use an ideal component of such a strategy, allowing users to interact with constructs such as XML-Schema types and structured documents as essential components of the language itself.

Composing Schemas

Suppose one prospective component of a solution architecture consists of a Web service for managing business development activities. A quick round of brainstorming reveals that one of the most commonly used domain objects is a "person," which can in turn be specialized to represent contacts, internal bizdev specialists, and so on. Since we're employing Immersive SOA, the next step is to quickly draft a person type using Tectonic.

Just like a schema definition composed using XML,

```
schema bizdev
   "http://www.proto.com/schemas/bizdev"
 type person
  fname;
  lname;
  mid;
 end person
end bizdev
write bizdev;
```

our schema definition begins with the word "schema." At this point, it consists of a single type, named "person," defined by the word "type" and a semicolon-delimited list of element names. The last line of the code block expresses the schema in document form. Both schemas defined in Tectonic and that are imported can be used for validation and instancing. Output is as follows.

```
<xs:schema ...>
 <xs:complexType name="person" >
  <xs:sequence>
   <xs:element type="xs:string"
        name="fname" ></xs:element>
   <xs:element type="xs:string"
        name="lname" ></xs:element>
   <xs:element type="xs:string"
         name="mid" ></xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:schema>
```

As simple as this may seem, there is an even easier way to define our bizdev "person." Tectonic maintains a default schema used to house all constructs not placed in named schemas.

```
type person
  fname;
  lname;
  mid;
end person
```

At this point, it becomes interesting to take a look at an instance of the type. In Tectonic, this is achieved using the keyword "instance."

```
pi = instance person;
write(pi!);
```

The character '!' is known as the elaboration operator, or "bang" for short (the expression pi! is, for example, pronounced "pi bang"), and causes a variable's structured content to be expressed as a string. Output is as follows.

```
<person>
  <fname>string</fname>
  <lname>string</lname>
  <mid>string</mid>
</person>
```

Type instances can be validated, transformed, persisted, and used to invoke Web services or generate output.

Authoring Services

Once a round or two of data model refinements have taken place, the next logical step is generally construction of a rudimentary service prototype. Using Tectonic, this can often be accomplished in the space of a few minutes.

Cretaceous Software Inc.
7229 W Franklin Blvd.
Boise, ID 83709
(208) 362-9625
www.cretaceousoftware.com

This example provides the first cut of a service to be used for maintaining a collection of business development manager profiles. The service, "BDMaint," starts out with a single operation "GetBDManager," to be used for obtaining information pertaining to a particular manager.

```
service BDMaint
  operation GetBDManager
   in(placeholder);
   out bdm format bd-manager;

   bdm = instance bd-manager;
  end GetBDManager
 end BDMaint
```

By using the statement

```
        out bdm format bd-manager;
```

to declare the output message, we indicate that it should be formatted using the "bd-manager" type. A single line of code...

```
        bdm = instance bd-manager;
```

... defines the operation's behavior, setting the output message content to a "blank" instance of "bd-manager."

These seven lines of code are sufficient to completely define the service. There's no framework data or code files linked or modified, no autogenerated source files, no manifests or descriptors or other deployment artifacts, no dependencies of any kind – nothing but the small, self-contained block of code shown in these examples. This code block could be IMed from one team member, who could then paste it directly into a blank Delve code page and immediately execute it.

Services defined in Tectonic are true SOAP Web services, and can be remotely invoked in the same manner as any other service, but they may also be invoked directly, in the same code blocks in which they are defined.

```
invoke BDMaint/GetBDManager
 response mgr;
write("\n\nManager:\n-------------\n");
write(mgr!);
```

The operation is functional but simplistic. We'll enhance it by: 1) having it accept an input parameter, something upon which to select a specific manager representation, and 2) filling in the response message with distinct values, overriding the defaults produced by instancing.

```
service BDMaint
 operation GetBDManager
  in(man-id-to-match);
  out bdm format bd-manager;
  bdm = instance bd-manager;
  bdm/fname = "Harry";
  bdm/lname = "Bendis";
  bdm/mid = "K";
  bdm/age = 42;
  bdm/affiliation =
   "Financial institutions, like UBank";
  bdm/@man-id = man-id-to-match;
  bdm/@title =
   "Business Development Principle";
  bdm/report/fname = "Alice";
  bdm/report/lname = "Campbell";
  bdm/report/mid = "F";
  bdm/report/age = 23;
 end GetBDManager
end BDMaint

invoke BDMaint/GetBDManager(A2334)
 response mgr;
write("\n\nfull name is " +
mgr/Body/bd-manager/fname/text()
 + " " + mgr/Body/bd-manager/mid/text() +
 " " + mgr/Body/bd-manager/lname/text());
```

Generating Sample Data

Prototyping exercises are often well served by the ability to readily compose significant amounts of sample XML. Tectonic includes a feature that serves this purpose in the form of the "executable element," a functional entity dedicated to the generation of XML data.

```
element(fn, ln, mi, ag, eid, af, loc, mid, tl)
   bd-man-gen
 tagname "bd-manager";
 elementi fname tnwrite(fn); end fname
 elementi lname tnwrite(ln); end lname
 elementi mid tnwrite(mi); end mid
 elementi age tnwrite(ag); end age
 elementi emp-id-num tnwrite(eid); end emp-id-num
 attribute man-id = mid;
 attribute title = tl;
end bd-man-gen
```

The executable element "bd-man-gen," is used to create a bizdev manager profile instance.

Let's suppose an architect wants to expand the BDMaint service to include an operation used to obtain an entire list of business development managers. In an early prototype, the operation's body consists of a single line of code,

```
bdms = bd-men();
```

setting the output message content to the result of calling an executable element that looks something like this:

```
element bd-men
 tagname "bd-mgr-collection";
 bd-man-gen("Sally", "Denton", "V",
   51, 8892,"Aerospace, Southern CA", "Los Angeles",
   "C3323", "Senior Bizdev Manager");
 bd-man-gen("Daniel", "Broward", "T", 33, 2332,
  "IBM Global Services, Cisco",
  "Los Angeles", "N23",
 "Business Development Manager");
 end bd-men
```

To learn more about Cretaceous Software and SOA fast prototyping, see http://www.cretaceoussoftware.com/collaboration.html.

SOA Software Service Manager 5.0 + Workbench 5.0 Product Review

Platform-independence, adapting to client consumption and service provider needs, and situational awareness

WRITTEN BY **PAUL O'CONNOR**

Many enterprises are currently reorganizing their people, processes, and technology around services. A few are holistically revamping their enterprise architectures around SOA and embarking on roadmaps to achieve grandiose business goals.

Far more enterprises are trying to deal with the unexpected emergence of services and service integration requirements resulting from platform upgrades, portal mashups, ESBs, and service-oriented business requirements such as external partner federation. No matter how they come to SOA, enterprises quickly realize that they need to manage and govern services. When these enterprises look out over the SOA governance landscape they see a jungle of registries, ESBs, Web services management systems, XML gateways, legacy identity and access management systems, ever-extending platforms, governance interoperability standards, and raw management standards. Knowing that "governance" is an almost primal need for IT management, technology marketing regimes have tried to fill the void with their brands. The result has been a great deal of confusion. It was with this state of affairs in mind that I recently had a chance to take a close look at SOA Software's Service Manager 5.0 and Workbench 5.0. I approached the run through of the product from the perspective of an enterprise architect tasked with governing and managing a Web services integration environment. This is a very common use case in my experience and falls squarely into the second type of services adoption pattern mentioned above that focuses more on the "nuts and bolts" of governance. Before proceeding, I think it is essential to state a set of "nuts and bolts" governance requirements:

- **Cradle-to-grave management of the service lifecycle**
  - Making sure the right service is available at the right time with a compliant interface description and with the right governance policies
  - Management of the approvals processes for each stage of the service lifecycle with custom workflow
  - Service/schema versioning support
  - Service development artifact repository and traceability
  - Collaboration environment for service stakeholders
- **Seamless registry/repository integration**
  - Intuitive user interface
  - Customizable roles/responsibilities, organizations, and views
- **Discovery and value governance**
  - Registry, search, and reuse enablement
  - Client collaboration and demand management
- **End-to-end security policy and trust management**
  - Authentication, authorization, encryption, non-repudiation
  - Access control policy via legacy I&AM system integration, as well as integration with emerging fine-grained access control systems
  - Integration with XML firewalls
  - First-mile and last-mile security policy enforcement
  - Audit logging
  - Integrated PKI management to support federation
- **Runtime management, monitoring, and SLA enforcement**
  - Monitoring, metrics collection and storage, alerting, problem root cause analysis
  - Custom dashboard creation
  - Unknown service discovery and management
  - Seamless SLA and contract management, intelligent routing and rejection based on SLAs
- **Mediation**
  - Semantic mediation policies
  - Technical mediation of transport protocols, message styles and exchange patterns, security token formats, reliable messaging styles

I have purposely left out other aspects that are sometimes included in the broader sense of SOA governance, e.g., business process analysis/modeling, asset management, portfolio management, information modeling, etc. Instead, the focus is on "nuts and bolts" governance: a set of functions that are common to almost every conceivable SOA, but nowhere near enough for some. I was glad to get a chance to put SOA Software's integrated registry/repository/management tool through its paces and found its value proposition to be quite compelling against this "nuts and bolts" governance requirements set.

Background on SOA Software

I started tracking SOA Software last year when it acquired Blue Titan and its mediation technology to go along with an established runtime governance and management tool. At that point it was apparent that they had a roadmap to get to an interesting point in

the space. Then they announced Workbench at the end of 2006, which is a design-time governance tool that completes the end-to-end governance and management cycle. Workbench and Service Manager are licensed separately; both include a registry/metadata-repository that they share when working in concert to provide a single participant experience. But the best part of SOA Software's vision, in my opinion, is that they are platform independent in that they include management/enforcement agents for every platform you can think of, every identity and access management system you might own, and both .NET and Java (for policy delegation out of code), and stand-alone mediation points for virtualization. For anyone doing governance on an enterprise, line-of-business, and likely even a departmental scale, this is a must. Governance must be adaptive and cannot entail re-platforming services. With this in mind, you might want to take a close look at SOA Software as you move down your own governance path.

Architecture and Installation

Service Manager and Workbench share a common administration server, which comprises a set of five individual processes working in concert with redundancy for fail-over. The admin server comprises the registry, repository, Web admin interface, and enforcement network management server. You can also reuse any existing UDDIv3 registry. Then there is a network of management points in the form of agents and stand-alone intermediaries. As previously mentioned, agents are available for virtually every conceivable platform service container, including all the app servers, ESBs I am aware of (except the open source ones), and the Microsoft stack as well. Installation was a snap and entailed little more than identifying an instance of one of the supported databases (Oracle 9i and above, MS-SQLServer 2000 and above, or DB2 v8.2) and supplying an admin username and password.

Initial Configuration and Workbench View

Figure 1 illustrates the management console browser app, which is a standard portal interface. Like any portal, the management console is personalized to each role and each organization. Notice the activity tabs across the top of the console. These switch the focus between the major focal points in the tool. The "Workbench" tab reflects the design view of the tool (separately licensed as mentioned earlier) and is where developers and administrators who collaborate on service life cycle, versioning, and workflow spend the majority of their time. Speaking of collaboration, one neat feature of the tool is its collaboration space associated with each service. Each service has its own forum in which developers, administrators, and architects can collaborate on service details. Coupled with the ability to attach unstructured data like design docs, UML diagrams, test plans, and the like, along with workflow history, traceability and process adherence is automated to a large degree. This can really be looked at as a CMMI enabler and streamlining vehicle.

On the left side of the Workbench view is the registry navigator. This is where the tried/true enterprise taxonomy is configured. In the example, I have configured a fictitious enterprise called "ACME Financial," as well as an "Account Services" service provider sub org (which hosts the "AccountManagerService" in view) and an "Account Services Clients" sub org that ostensibly consumes the service internally. Then there is an external partner organization called

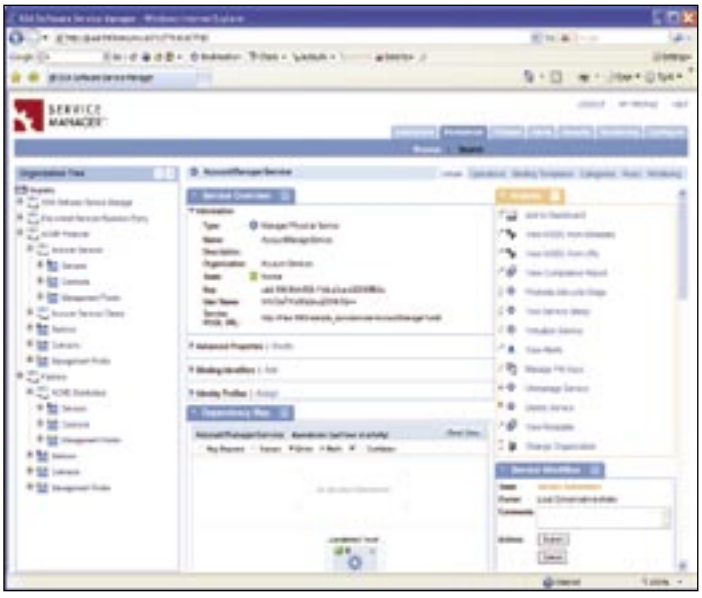


Figure 1:

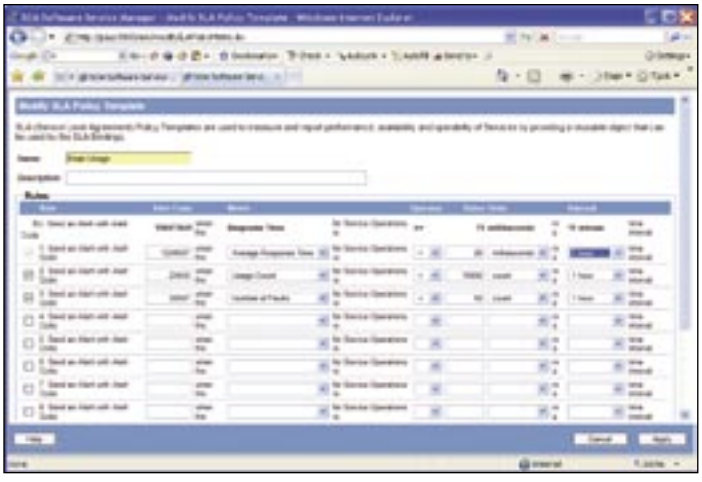


Figure 2:

"ACME Distributors" that also consumes the account manager service. Notice that each organization and sub-organization has a "Services," "Contracts," and "Management Points" folder to contain associated artifacts. We will shortly see that it is the contracting process in the tool that generates much of its power. However, before moving on let me point out some of the interesting actions on the right side of the view.

There are three types of services that the system knows about: registry services, physical services, and virtual services. Registry services are those that are somewhere in the design process but not yet necessarily managed in a container. You don't even need a WSDL to create a registry service, just a thought that a service of some sort is needed. Once it is entered into the registry, workflow and collaboration ensues toward the ultimate provisioning of a production service in a managed container. Along the way, compliance policies are applied to service artifacts as configured in policy templates associated with each life-cycle stage in the process, which is itself configurable. There are stock compliance policies that ship with

the tool, like WS-I Basic Profile 1.1 compliance. Custom polices are added as XQuery expressions against service artifacts like WSDLs.

Registry services that are managed at some point become physical services. Physical services can also be created immediately if the container they are to run on has an agent. Just work your way through a simple wizard to associate a WSDL and management point and your service will be under management, and at the beginning of your established publishing cycle. Virtual services are hosted on stand-along management points and are used, not surprisingly, for mediation. The tool has compelling mediation capabilities including semantic mediation (XSLT transforms), transport mediation (SOAP to MQ, BEA-JMS, etc.), message styles (SOAP to

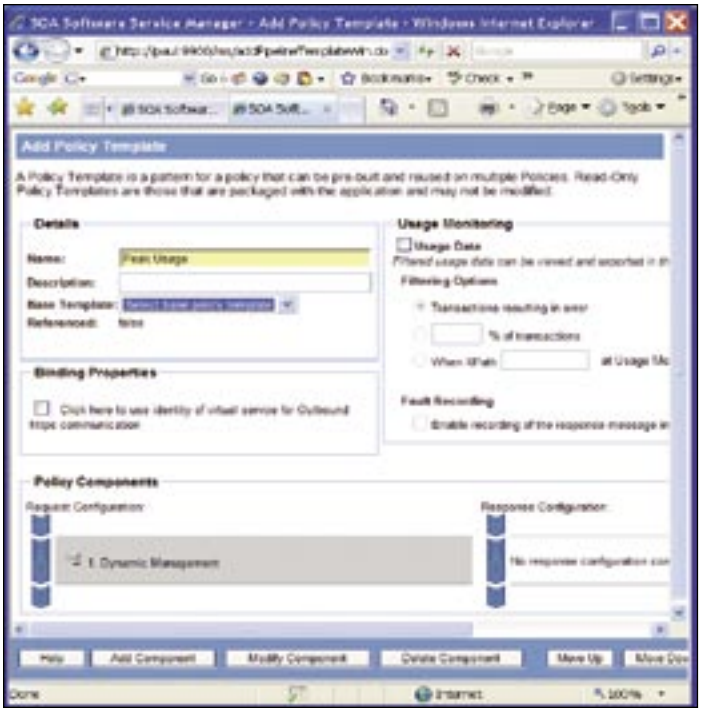


Figure 3:

POX, RSS, and REST), message exchange patterns (synch/asynch), reliable messaging formats with correlation, and security token formats (like Kerberos to SAML). With the addition of the Blue Titan mediation platform, the tool is able to mediate across a dizzying array of standards versions and differing standards interpretation impedances. I would go so far as to say that it is very difficult to effectively govern without a pretty far-reaching set of mediation capabilities.

Contracts and SLAs

The heart and soul of the tool is its contract and SLA management functionality. Contracts are active in the sense that they govern the utilization of services at runtime and track the relationship between provider and consumer from relationship inception through versioning and SLA changes. This is the fulfillment of the closed-loop governance vision and SOA Software has trademarked the contracting process in the tool, calling it an "Active Contract"TM. Contracts are negotiated between consuming organizations and service providers and can be applied to services down to the operation level. The tool provides configurable workflow around negotiation of service capabilities (highlighting the importance of mediation again in the context of being able to meet client demands), policies, and SLAs. Figure 2 illustrates a common SLA policy template being configured in the tool and which tracks average response time, usage count, and fault count. Figure 3 illustrates the creation of a dynamic management policy with a service consumer throttling rule. A contract could be negotiated that would establish an SLA based on the template created and a dynamic management rule that would throttle access to the service for client requests from the consuming organization under the contract when the SLA was violated and the associated alert issued.

Monitoring and Auditing

The tool provides excellent monitoring and auditing functionality. Alerts can be set up for performance and policy violation metrics, tailored to different consumers, channels, and thresholds. Dashboards are customizable to services, service groups, organizations, and roles. Audit trails provide a comprehensive view of policy violations and actions taken. The integrated view of service performance with respect to contracted policies in the context of the service life cycle and position in the enterprise taxonomy leads to great situational awareness. This is the beauty of closed-loop governance.

Other Features

In the interest of time and space, I will lump together some of the remaining features of the product that I can think of. First, I should mention rogue/unknown service discovery. Embedded management points that spot service traffic in their containers that are not in the registry cause those services to appear as "discovered services" in the Workbench view. Policies can be configured to deal with such services, e.g., block them, alert administrators, etc. Also, there is a whole aspect to the tool that deals with value governance. Value governance is a phrase that refers to being able to manage service consumption in business terms. By assigning arbitrary value to service transactions, even basing the value on the message context, e.g., a dollar value indicated by an XPath expression into the message SOAP Body, a value model can be created around strategic services. Such a model can support a pay for play charging model, which certain businesses (think SaaS) desire. The one last feature that bears mentioning is the Java and .NET delegation APIs. These APIs allow service clients to delegate service calls to a delegator, which interacts with the tools's policy repository to create a compliant service call based on the security policies in force for the service.

Conclusion

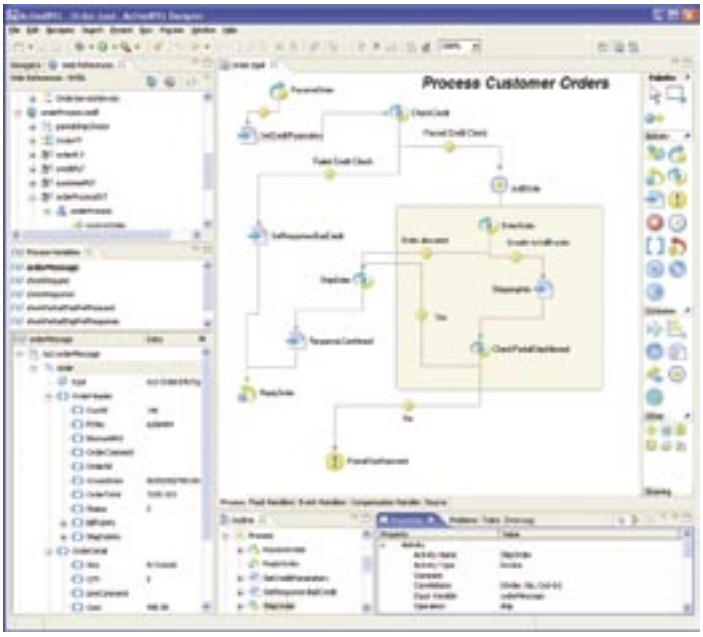
Governance is an overloaded word in SOA, and indeed an overloaded concept. Most enterprises that I am aware of would do well establishing a subset of the broader governance aspects tailored to their immediate needs – "nuts and bolts governance." To be effective, governance must be platform independent, adaptive to client consumption and service provider needs, and closed-loop in the sense that situational awareness is provided. I think SOA Software's recent revision of Service Manager (version 5.0) coupled with Workbench 5.0 serves these functions in fine fashion. ■

About the Author

Paul O'Connor is SOA practice director and chief SOA architect for e-brilliance LLC (a leading NE SOA consultancy), and is currently doing major SOA architecture and implementations for Fortune 100 clients across the U.S. Previously he was chief architect for Damascus Road Systems, specializing in security architecture.

poconnor@e-brilliance.com

Get Started with BPEL 2.0



Build next-generation SOA applications with the leader in BPEL technologies

Download BPEL tooling & server software today

active-endpoints.com/soa

BPEL consulting and training.
BPEL design tools, servers and source code for Eclipse, Apache Tomcat, JBoss, WebSphere, WebLogic, BizTalk and Microsoft .NET.

activeBPEL

active endpoints

Altova® XMLSpy® 2007 – The industry standard XML development environment.

ALTOVA® xmlspy® 2007

Immerse yourself in XML intelligence

Dive into XMLSpy® 2007, and take XML development to greater depths.

Spied in Version 2007 Release 3:

- Deep integration with IBM® DB2® 9: pureXML data server
- Support for new Microsoft® Office document format – Open XML
- New database query window supporting SQL, SQL/XML, and XQuery
- New intelligent CSS editor with context-sensitive entry helpers

Altova® XMLSpy, the industry standard XML development environment, is vital for modeling, editing, transforming, and debugging XML applications. Breathe life into your plans with the world's leading XML editor, the original graphical schema designer, a code generator, file converters, debuggers, profilers, full database integration, support for XSLT, XPath, XQuery, WSDL, SOAP, and a trove of treasured XML utilities and usability aides. Become a markup mastermind!
Download XMLSpy 2007 today:
www.altova.com

XMLSpy is also available as part of the value-packed Altova MissionKit™ software bundle.

ALTOVA®