

OUR NAME HAS CHANGED!
FORMERLY SOA WEB SERVICES JOURNAL

FOR ALL THE LATEST NEWS
AND INFORMATION
VISIT:

www.SOA.SYS-CON.com

SOA WORLDTM

M A G A Z I N E

MARCH 2007 / VOLUME: 7 ISSUE 3

— KEY TO ACHIEVING THE PLUG-AND-PLAY GOALS OF SOA —

SECURITY IN A SOA

RAJIV GUPTA 16

3 The Most Underestimated and Undervalued Aspect of SOA

SEAN RHODY

4 Understanding SOA Architectures and Models

DAVID S. LINTHICUM

6 Scaling Enterprise SOA Deployments

ATUL SAINI

12 A Review of SOAtest 5.0

KYLE GABHART

20 How To Build Flatter BPEL Processes

MICHAEL HAVEY

28 The Indispensable SOA

MICHAEL LIEBOW

30 FastSOA Patterns

FRANK COHEN



PLEASE DISPLAY UNTIL APRIL 31, 2007

\$6.99US \$7.99CAN



SOA WORLD 2007
CONFERENCE & EXPO

THE SOA EVENT
OF THE YEAR!

June 25-27, 2007
New York City



SOAEOCONFERENCE.SYS-CON.COM

Give your data direction

Link up with MapForce® 2007,
and exchange data with ease.

Spied in MapForce 2007:

- Support for Web services as sources, targets, or data processing functions in data integration projects
- Advanced capability for refactoring data mappings
- Tighter integration with Microsoft® Visual Studio® .NET

Altova MapForce 2007, the award-winning data integration and Web services implementation tool, makes it easy to exchange data between XML, database, flat file, EDI and/or Web services formats and to map data to WSDL operations. Simply drag connecting lines from data sources to targets and drop in data-processing functions. MapForce converts data on-the-fly or auto-generates program code in XSLT 1.0/2.0, XQuery, Java, C++, or C# for royalty free use in your data integration and Web services applications. Get connected!

Download MapForce® 2007 today: www.altova.com

MapForce is also available as part of
the acclaimed Altova MissionKit.

Supports official new
W3C standards:
XSLT 2.0, XPath 2.0
& XQuery!

INTERNATIONAL ADVISORY BOARD

Andrew Astor, David Chappell, Graham Glass, Tyson Hartman,
Paul Lipton, Anne Thomas Manes, Norbert Mikula, George Paolini,
James Phillips, Simon Phipps, Mark Potts, Martin Wolf

TECHNICAL ADVISORY BOARD

JP Morgenthal, Andy Roberts, Michael A. Sick, Simeon Simeonov

EDITORIAL

Editor-in-Chief

Sean Rhody sean@sys-con.com

XML Editor

Hitesh Seth

Industry Editor

Norbert Mikula norbert@sys-con.com

Product Review Editor

Brian Barbash bbarbash@sys-con.com

.NET Editor

Dave Rader davidr@fusiontech.com

Security Editor

Michael Mosher wjssecurity@sys-con.com

Research Editor

Bahadir Karuv, Ph.D. Bahadir@sys-con.com

Technical Editors

Andrew Astor andy@enterprisedb.com

David Chappell chappell@sonicsoftware.com

Anne Thomas Manes anne@manes.net

Mike Sick msick@sys-con.com

Michael Wacey mwacey@csc.com

International Technical Editor

Ajit Sagar ajitsagar@sys-con.com

Executive Editor

Nancy Valentine nancy@sys-con.com

PRODUCTION

ART DIRECTOR

Alex Botero alex@sys-con.com

ASSOCIATE ART DIRECTORS

Abraham Addo abraham@sys-con.com

Louis F. Cuffari louis@sys-con.com

Tami Beatty tami@sys-con.com

EDITORIAL OFFICES

SYS-CON MEDIA

577 CHESTNUT RIDGE ROAD, WOODCLIFF LAKE, NJ 07677

TELEPHONE: 201 802-3000 FAX: 201 782-9637

WEB SERVICES JOURNAL (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:

WEB SERVICES JOURNAL, SYS-CON Publications, Inc.

577 Chestnut Ridge Road, Woodcliff Lake, NJ 07677

©COPYRIGHT

Copyright © 2007 by SYS-CON Publications, Inc. All rights reserved. No part of this publication may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopy or any information storage and retrieval system without written permission. For promotional reprints, contact reprint coordinator, SYS-CON Publications, Inc., reserves the right to revise, republish, and authorize its readers to use the articles submitted for publication. All brand and product names used on these pages are trade names, service marks, or trademarks of their respective companies. SYS-CON Publications, Inc., is not affiliated with the companies or products covered in Web Services Journal.

Who's in Charge Here?

The most underestimated and undervalued aspect of SOA



WRITTEN BY SEAN RHODY

You know, I love an election year. The drama, the emotion, the positioning, it all makes me think about running for office myself – or at least going through the motions to generate a large war chest that I can dip into (I AM from New Jersey, it's a time-honored tradition). Oh, wait a minute this isn't an election year. Not that you'd know it from the slew of politicians tossing their hats in the ring. I guess it's a good idea to get in the race early if you're aiming for the brass ring.

Service Oriented Architecture is a complicated endeavor: it provides great flexibility in implementing business functionality at the cost of additional management and oversight.

Some part of that additional management responsibility falls on the shoulders of a new breed of software – the composite application, which is made up of services combined into processes. What used to take place in a single application now takes place across a network where messages inform services and processes control functionality.

But there's another set of management tasks that don't have software realization – they fall squarely in the lap of human management. We usually refer to these tasks as SOA governance.

Governance is all the tasks that revolve around managing business processes in a service environment except possibly those that actually get implemented in software. Even those have a definition phase that relies on human interaction and approval processes for final implementation.

Defining services, and composing business processes out of services, isn't simple. It requires cooperation among multiple groups in most cases and usually involves the coordination of competing priorities. For example, one organization I worked with in the past had over 100 separate ways to calculate the details of a customer account. Part of this was simply systems history as methods for determining the account evolved over time as systems were added. Another part of the puzzle revolved around the fact that different groups in the organization had different needs and requirements for systems relating to the customer account. So each individual fiefdom created its own slightly different version of the mechanism for determining the account. The end result – a nightmare of similar methods that conceptually all did the same thing – but were divergent in one or more ways from one another. Rationalizing a single method of evaluating the value of a client account from over a hundred variants is a challenging exercise, but it's ultimately achievable. But only if SOA governance provides a means of resolving issues.

And these issues, while not technical, can prove to be the most serious threat to actually implementing a Service Oriented Architecture. Because they're the ones that can stop a project in its tracks, cancel its funding, and subvert technical goals for business or political reasons.

Take the client account, for example. While it may make absolute sense from a technical perspective to have a single view of the customer and his accounts, it may not make political sense to the parties that currently have control of the fragmented solitary views of that same customer account. Governance under such circumstances is as much about driving political and organizational change as it is about determining the best technical approach to software development in a services environment.

This is the most underestimated and undervalued aspect of Service Oriented Architecture – its capacity to become a change agent in an organization based on the need for consolidating and rationalizing the service catalog. With proper planning and the right backing, the case for change can be a powerful weapon for the entire organization. Without such backing, governance issues and organizational politics can easily defeat an SOA effort regardless of the technical feasibility of the implementation. ■



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

Understanding SOA Architectures and Models

The SOA reference model PART1

WRITTEN BY DAVID S. LINTHICUM

A few people who have been reading my blog and this column, and listening to my podcast, as well as reading other SOA blogs and articles, have become a bit confused pertaining to the notions of:

- SOA Reference Model(s)
- SOA Reference Architecture(s)

And how all of this works and plays with

- Enterprise Architecture

I spent a few hours of my weekend attempting to research and define these concepts a bit better, in essence, taking everyone's opinions and normalizing them so they make better sense. What I found were many of the same notions, defined differently, but all attempting to solve the same problems. Seems to be a common theme within the world of SOA, but I digress.

Indeed, there are many definitions for the above concepts (not those terms specifically) that are now being defined by guys like me, standards organizations such as OASIS and the Open Group, and vendors such as IBM, BEA, WebMethods, and TIBCO. Sometimes they align; most of the time they don't.

Who's right? I'm not sure this is a matter of right and wrong, but perhaps it's time we come up with some common definitions and shared vision, as I've been saying. I do think we are moving in that direction, albeit slowly, and I think that just agreeing on semantics will be a huge accomplishment in this emerging space.

I'll tell you what I know; you can make your own choices. Let's begin by exploring the concepts being put forth by the standards organizations. This will be backing up a bit from the recent posts and columns that have basically restated and commented on the news; I'm attempting to provide better context. Also, to avoid the many e-mails and comments I get when writing about this stuff from those who write the standards and define these terms, I'm going to be quoting more than I typically do. Also, I engaged an insider for fact-checking, JPL's Jeffrey A. Estefan, who is authoring the standards. Of course, I have to add my 2 cents. Here goes...

SOA Reference Model

Now let's explore the concept of the SOA Reference Model, also a product of OASIS, and how the SOA Reference Model and SOA Reference Architecture relate to each other. From the OASIS Reference Model for Service-Oriented Architecture.

*A **reference model** is an abstract framework for understanding significant relationships among the entities of some environment. It enables the development of specific reference or concrete architectures using consistent standards or specifications supporting that environment. A reference model consists of a minimal set of unifying concepts, axioms and relationships within a particular problem domain, and is independent of specific standards, technologies, implementations, or other concrete details.*

— continued on page 27

About the Author

Dave is the CEO of the Linthicum Group, LLC, (www.linthicumgroup.com) a consulting organization dedicated to excellence in SOA product development, implementation, and strategy. david@linthicumgroup.com

CORPORATE

President and CEO

Fuat Kircaali fuat@sys-con.com

Group Publisher

Jeremy Geelan jeremy@sys-con.com

ADVERTISING

Senior VP, Sales & Marketing

Carmen Gonzalez carmen@sys-con.com

VP, Sales & Marketing

Miles Silverman miles@sys-con.com

Advertising Sales Director

Megan Mussa megan@sys-con.com

Associate Sales Managers

Corinna Melcon corinna@sys-con.com

SYS-CON EVENTS

Event Manager

Lauren Orsi lauren@sys-con.com

CUSTOMER RELATIONS

Circulation Service Coordinator

Edna Earle Russell edna@sys-con.com

SYS-CON.COM

VP 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

SYS-CON Publications, Inc., reserves the right to revise, republish and authorize its readers to use the articles submitted for publication.

Fiorano SOA™ 2006

The Quickest path to an SOA

- ✕ FioranoMQ™ 2006 – world's fastest, most scalable JMS
- ✕ Fiorano ESB™ 2006 – CAD/CAM for distributed applications
- ✕ Fiorano BPEL Server – simplifying business process orchestration
- ✕ Fiorano Tools – BPEL Studio, Mapper, FEPO, etc
- ✕ Fiorano Components – 60+ pre-built adapters



Benefits

- ✕ Adherence to popular industry standards - JMS, COM, .NET, JCA, JMX, BPEL, SOAP, etc.
- ✕ Multi-language, Multi-platform, Multi-protocol
- ✕ Unmatched Scalability and High Performance
- ✕ Quick, Measurable ROI

Download your copy of Fiorano today!

www.fiorano.com/downloadsoa

Fiorano®
Enabling Change at the speed of thought

Scaling Enterprise SOA Deployments

The benefits of a service-grid architecture powering next-generation SOA deployments

WRITTEN BY ATUL SAINI

➤ The first wave of integrating storage, computing, and networking hardware helped businesses move from client/server to Internet-based peer-to-peer networks. A second wave of integrating applications on top of the hardware infrastructure promised to deliver unprecedented economies of scale. In today's enterprise IT model, applications exposed as services need to be integrated seamlessly with other applications distributed across the network to generate the best operational efficiencies. Messaging-oriented middleware is at the heart of enabling seamless or "effortless" integration between a business's core assets: its applications and data residing on the network.

However, integrating multi-vendor applications with diverse infrastructure and legacy applications is a daunting task. In a recent report—a leading analyst reports that Global 3500 firms will spend an average of \$6.4 million each in 2007 on Service Oriented Integration and process management budgets, and less than 35% of the projects come in on time and on budget. So why are SOA project schedules so unpredictable? Incumbent first-generation SOA solutions consist of a patchwork of point products that are struggling to meet the performance, scalability, flexibility, and security demanded by modern

distributed business applications. In contrast to an age of irrational exuberance, throwing more hardware and customized software at the problem is no longer a viable option for most businesses. Since early 2003, the Enterprise Service Bus (ESB) has emerged as a standards-based Services Orchestration Platform, typically built on a distributed peer-to-peer service-grid architecture that enables high-performance, scalable, and affordable SOA solutions.

A distributed Enterprise Service Grid typically consists of the following server components:

- **Enterprise Server (ES):** The ES forms a centralized repository for security policies, business process configuration and logs, monitoring, debugging facilities, and reusable services
- **Peer Servers (PS):** These lightweight servers can be installed on handheld devices as easily as on back-end enterprise servers. With a local store-and-forward service, PSs enable a peer-to-peer data transport thereby generating huge economies of scale
- **MQServer:** A standards-based JMS messaging server that supports an arbitrary number of PSs communicating with an arbitrary number of ESs—thereby enabling a brokered peer-to-peer services orchestration platform in which control information flows through a centralized server (the ES) while data flows concurrently between connected Peer Servers (PSs).
- **Services Orchestration Tools:** A set of developer-centric tools that enable the composition, governance, monitoring, debugging, and creation of reusable services.
- **Built-in Adapters:** ESBs typically include hundreds of available pre-built adapters for packaged as well as legacy applications.

The distributed architecture of a modern Enterprise Service Grid provides ease of use and real-time responses from integrated services as discussed in the rest of this article.

Scalable Performance

Applications and data are the lifeblood of a business. Faced with dynamically changing customer, partner, and supplier requirements, businesses need to integrate a large number of applications – exposed as services – in multiple data formats across dispersed geographic locations. Traditional middleware integration solutions consist of Business Process GUIs hooked into a distributed services layer as shown below.

An arbitrary number of standalone applications can be integrated to represent a workflow that could model, for instance, a financial transaction processing package. The hub-and-spoke messaging middleware also uses standards-based protocols (TCP/IP, HTTP, and others) to process and manage the messages supporting a set of workflows.

All messaging traffic between distributed services consisting of control and data packets must traverse the HUB – also referred to as an Integration Server (IS). As traffic requirements scale, the IS becomes a performance bottleneck and a single point of failure. IS clusters can help create redundancies that increase availability – at a high cost of increased complexity and money. Most IS clusters rely on multicast mechanisms to relay messages. The multicast mechanism is very inefficient in terms of bandwidth use, especially when delivering certified/guaranteed messages – as demonstrated in several benchmarks. While composing services into composite applications remains the key value of an SOA solution, underlying hub-and-spoke architectures are weak in delivering scalable performance and guaranteed message delivery at high rates.

The Brokered Peer-to-Peer architecture of a modern distributed Service-Grid platform provides near-linear scalable performance at highly affordable costs. The distributed Service-Grid architecture splits up data traffic among an arbitrary number of lightweight peer servers and controls the traffic with enterprise servers supporting all centralized functions. An Enterprise Service Grid Network consists of Peer Servers (PS) connected to a set of Enterprise Servers (ES) via a JMS messaging backbone as shown in Figure 2.

As seen in Figure 2, the distributed peer-to-peer architecture of an Enterprise Service Grid provides several benefits:

- 1) **Store and Forward.** Peer Servers (at \$1k/each) with their local store/forward mechanisms and a micro-messaging server enable fine-grain control over the performance scalability of data traffic
- 2) **Enterprise-Class Backbone.** JMS-compliant MQ servers typically support over 10,000 messages/sec and a large number of concurrent connections thereby providing a very scalable enterprise-class messaging backbone. With a dual-redundant configuration that supports failover, the MQ backbone deploys an arbitrary number of PS (data traffic) and ES (control traffic) at very affordable costs. This provides a build-as-you-grow scalability model with no downtime or added complexities.
- 3) **Enterprise Servers (ES)** are constantly updated with all control traffic from any of the PS peers. The MQServer arbitrates between outgoing traffic from the ES to the PS servers and performs many-to-one de-multiplexing for this traffic. The JMS-compliant MQServer also typically supports sophisticated load-balancing

algorithms such as round robin, weighted round robin, and Intelligent Load Balancing (ILB) – to provide the efficient use of all PS and ES deployments in the system. The distributed Service-Grid architecture inherently provides for a highly scalable performance envelope. SOA architects can achieve very fine-grained control to match their performance requirements without trading off scalability or reliability goals as measured by the following parameters:

- Number of messages per second (aggregate throughput through the system)
- Size of the messages (5KB, 100KB, 1MB, 5MB)
- Number of concurrent applications/users supported
- Messaging Quality of Service (MQOS): delivery class (normal v/s guaranteed delivery)
- MQOS: Total Latency (total application-to-application latency via Integration Servers)

A Service Grid Peer Network consisting of two ES and a handful of PS servers can deliver on-par performance of 100-500 messages/sec at a fraction of the cost of traditional hub-and-spoke-based solutions as shown in Figure 3.

Peer-to-Peer Service Grids deliver unprecedented levels of throughput, capacity, and MQOS, easily outscoring the nearest competitors by a factor of 10x. When performance benchmarks are normalized to the

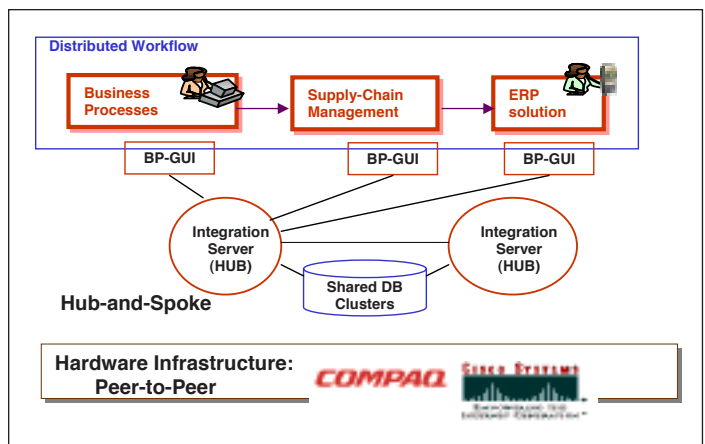


Figure 1: Traditional hub-and-spoke messaging architectures

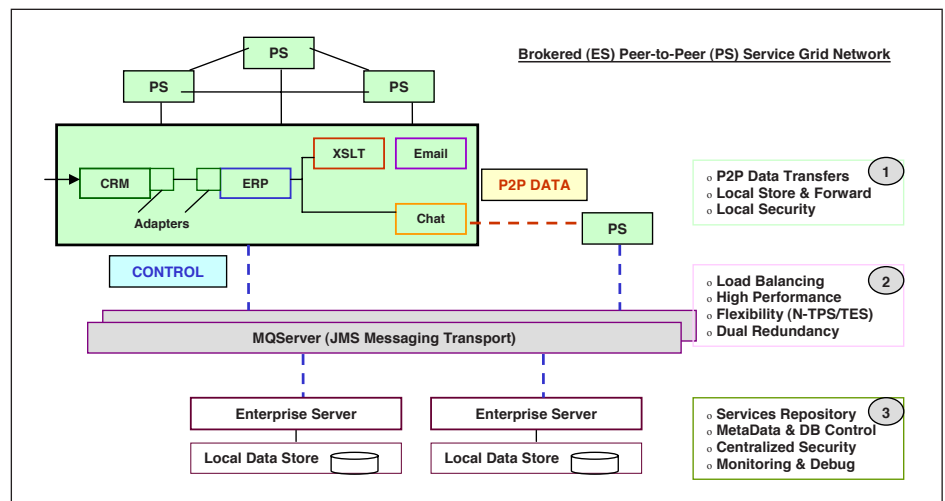


Figure 2: A ESB peer network consisting of servers (ES and PS) and MQ transport

cost of software (and in some cases, the minimum hardware required), a P2P Service Grid delivers even higher returns on investment in the performance benchmarks as seen in the plot in Figure 3.

- (1) Indicates a total cost of (\$1k/PS * 20 + \$25k/ES = \$45k)
- (2) Indicates a total cost of (\$1k/PS * 10 + \$25k/ES * 2 = \$60k) and provides a failover configuration in case of an ES failure.
- (3) Indicates an average cost of (\$300k x 3 x 2 = \$1,800k) with three Integration Servers to provide an on-par performance of 100-500 messages/sec.

High Availability

System-level high availability is a function of the availability of its components. While evaluating SOA platform solutions, it's imperative that each of the SOA solution components is designed for and enhances the overall availability ratings with complete failover capabilities:

- Distributed Processes (GUIs that help compose and deploy workflows, any local store-and-forward data queues and associated software, distributed hardware)
- Messaging backbone
- Adapters to various applications
- Service-level failover (It should be possible to relaunch a failed service on a different machine in the case of distributed architectures for business/Web Services)

The distributed P2P Service Grid architecture enables clear visibility at each of the component levels of an SOA infrastructure solution as listed above and shown in Figure 4.

The brokered peer-to-peer architecture of an Enterprise Service Grid enables any PS to designate another as its secondary or tertiary backup in case of failure. Mechanisms built into each PS monitor the health of its hardware and ensure that in case of hardware failure, all completed and pending transactions failover to the designated secondary transparent to the applications themselves.

Control traffic from a PS – for example, the creation and registration of a new service in the workflow – is immediately sent to multiple ESes via the MQ messaging backbone. Each ES has its own data store – unlike hub-and-spoke solutions that have shared data stores between Integration Servers. In case of an ES failure, a designated secondary ES transparently takes over its functionality, with no loss of transaction-level details.

The MQ messaging backbone that includes intelligent load balancing of messaging traffic and supports both the ES and PS instances, is itself built as a redundant system to support failover. Thus, the Enterprise Service Grid enables a self-healing network of peers that protect against any component-level failure.

Traditional hub-and-spoke architectures in contrast necessitate replicating expensive Integration Servers to achieve reliability through redundancy. Even the touted “Federated Integration Servers” from TIBCO or the “Distributed Peer-to-Peer” solutions from IBM rely on shared memory implementations that become a single point of failure. Increasing reliability raises costs and complexities dramatically, which in turn can adversely affect overall performance and scalabilities.

Experience with a variety of customer implementations shows that the Enterprise Service Grid's self-healing networks offer the highest levels of affordable reliability under various real-world operational scenarios.

Security

Messaging middleware is a critical component of any enterprise's overall security policy framework. Messages in transit between par-

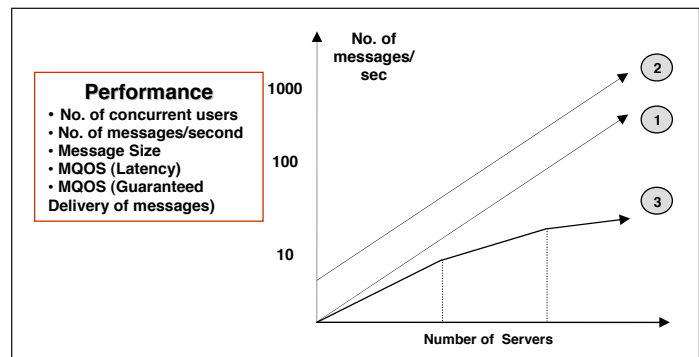


Figure 3: Scalable performance with an ESB versus traditional EAI solutions

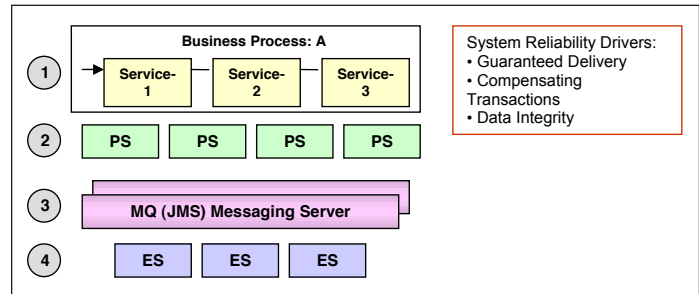


Figure 4: Enterprise Service Grid system components supporting overall high availability

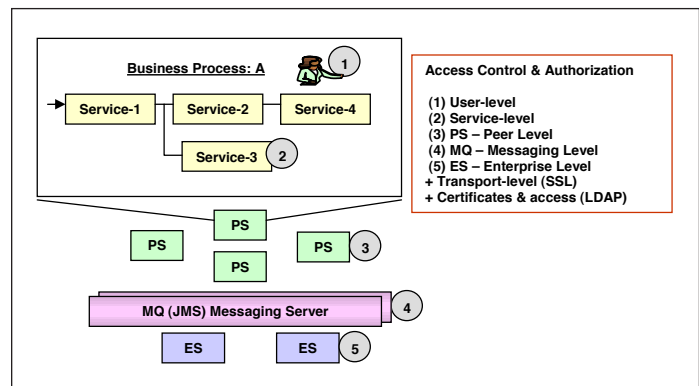


Figure 5: Easy integration and support of overall security policies with a Service Grid

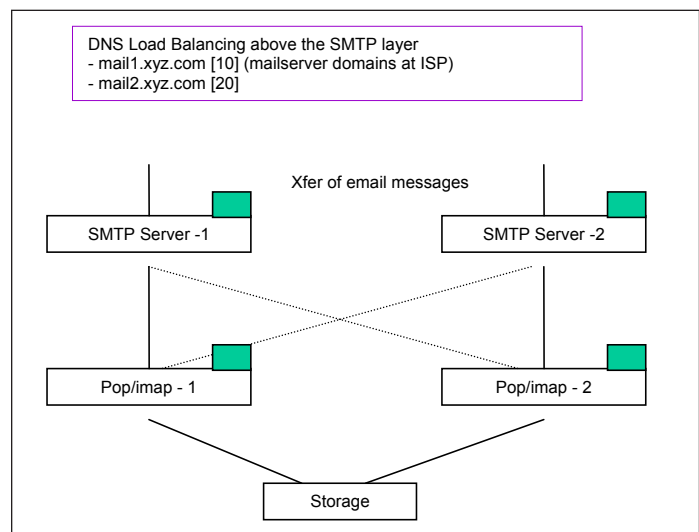


Figure 6: Distributed peer-to-peer IT asset management system

Achieve Point-and-Click SOA™

exteria
Business Automation Platform

*Get on the SOA
fast track!*

NEXT STOP: extentech.com

FREE Exteria Downloads

Demos and Whitepapers

We're addicted!

Now that we are going to output to the Web, ExtenXLS has saved us a tremendous amount of time and work."

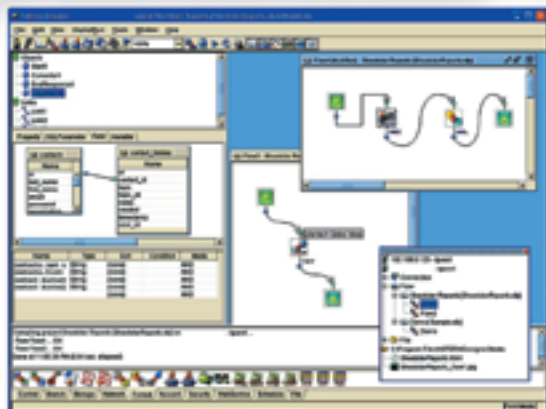
-Steve T, System Architect,
B2B Application Development
Pfizer, Inc.

High-Speed Productivity

Exteria is the ideal modeling tool and web services platform for rapid SOA implementations.



Model data flows graphically in the code-free designer then publish with a click to XLS, PDF, SOAP, XML, and more -- all without writing any code!



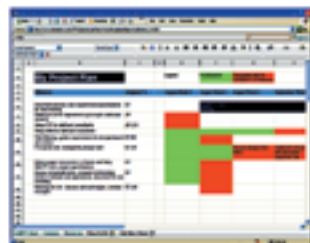
The Exteria data flow modeler features dozens of connectors including LDAP, SOAP & XML

Spreadsheet Automation

ROI is all about re-use. There is no need to re-code the business logic already available in spreadsheets. Leverage this asset by mapping data flows to formula cells and publishing them as RSS feeds, REST API and SOAP web services, or to a web spreadsheet.

works with:
sheetster

Integration with Sheetster.com web spreadsheets enables realtime data feeds and collaboration.



includes:

EXTENXLS 5
JAVA SPREADSHEET TOOLKIT

Includes the best
Java Spreadsheet
Automation available...
ExtenXLS 5.1!

Copyright 2007, Extentech Inc., Portions Copyright 2001-07 Infasteria Corporation. All Rights Reserved.

extentech
exceed expectations

sales@extentech.com
call 415.759.5292
<http://extentech.com/itsg>

Strategic Requirements	Service Grid: Architecture-Level Sustainable Advantages	Incumbent architecture-level weaknesses
Scalable Performance	Data: Peer-to-peer scalability within the Peer Servers (\$1k/each) Control: Real-time replication of centralized information in Enterprise Servers (\$25k/each) Messaging backbone: JMScompliant MQ with built-in loadbalancing and support for over 10000 msg/sec; amenable to be clustered as needed. NET: Fine-grained control over data and control messaging traffic with near-linear scalability in performance	Data and Control routed on a hub-and-spoke architecture. Even solutions claiming to be distributed architectures rely on shared memory semantics at the Integration server levels – thereby limiting scalability and performance.
High Availability	Self-healing ESB Network ensures a 24x7 level of availability built-into the distributed network at no additional cost.	Requires redundant Integration-servers adding to the costs and management/deployment complexities
Security	Role-based security with fine-grain access control at the user, workflow, host (machine hosting the workflow) and individual service levels	Weak security models, difficult to integrate into enterprise security policy frameworks
Extensibility	Integration is only one of the potential areas of application. Enterprise Service Grid's are being considered for solving fail-over issues in routine IT management tasks, as well as deploying a variety of new solutions within an enterprise via flexible composite applications.	Limited to Integration usage.

Table 1: Architecture-level benefits summary with an Enterprise Service Grid versus leading-edge SOA alternatives leading-edge SOA alternatives

ticipating publishers and subscribers need to be secure and tamper-proof. Messages stored in local queues need to be secured as well.

Messages once consumed don't have to be stored and should be purged from all databases that could be vulnerable to inspection by malicious agents. An Enterprise Service Grid can be easily integrated into a wide range of security policies as shown in Figure 5.

An Enterprise Service Grid's security and governance policy includes empowering security administrators to limit access and control to a business process at the user, service, and component level. Fine-grained security support enables a seamless security implementation inside and outside the corporate firewalls. Messages can be securely transported using HTTPS and SSL protocols (client- and server-side certificates and authentication supported). Security certificates can be stored redundantly in multiple Enterprise Servers (ES) across the Enterprise Service Grid network.

Extensibility

An SOA solution requires a vast array of reusable services and adapters. Adapters are software programs that deal with translating data stored in proprietary formats for applications such as PeopleSoft, Oracle DB, and SAP – into standards for data transport such as XML. Initiatives such as JCA and JBI (Java Business Integration) have enabled standards to demystify adapter issues. Incumbent integration solutions that build on hub-and-spoke architectures and adapters to integrate applications can only position themselves as extending with their adapters – deeper into the integration space. In contrast, integration is only one of several areas of application with an Enterprise Service Grid in an enterprise deployment. IT staff can write Enterprise Services easily in a wide variety of development languages (including Java, C, C++, and C# in most cases), and deploy these services over the Service-Grid network as intelligent software agents to assist in automating the management of several IT tasks.

Consider, for example, a set of SMTP and Pop/iMAP dual-redundant servers as shown in Figure 6. Incoming e-mails are stored in disk space on the SMTP servers. In the event of a disk failure, these e-mail messages need to be restored. Dual-redundant servers can provide for message backup; however, the mechanism with several scripts is process-intensive and a complexity overhead for IT staff. A more robust and easier alternative is to deploy PS agents on the SMTP servers and write service-based composite applications for failure recovery.

IT engineers can leverage the local store/forward of the Service Grid peer servers to save e-mail messages that aren't committed to a failed disk yet. Since the Service Grid peers are capable of real-time data transfers, these messages can be stored in the PS queues without being stored on the SMTP server disks. So the distributed PS network can be leveraged to implement failover less complexly. This simple idea can be extended to solve several failover issues throughout an IT-stack. If the SMTP servers in the example above were substituted with storage or networking devices, the Enterprise Service Grid can power sophisticated – and yet simple-to-use – IT management networks.

The Enterprise Service Grid's built-in security support at the service, PS, ES and transport levels enables its application in solving a wide variety of distributed problems. PS daemons installed on machines inside and outside corporate firewalls, as seen in Figure 6, can help with standards-based remote monitoring, deployment, and debugging tasks.

Mainstream collaborative applications can use the Service Grid as an integration and administration platform by building distributed application-specific services and adding to their services palette containing reusable services. The widespread adoption of a single platform and reusable services leads to dramatic savings in total cost of ownership (TCO) across enterprise solutions.

Summary

A leading analyst estimates that the Global 3500 will spend an average of \$6.4 million each on their SOA projects in 2007. Further, according to Gartner, 65% of these projects take longer and can cost more than the projected budgets. One of the key reasons for this unpredictability and expensive costs is the weak hub-and-spoke messaging architecture powering the SOA projects. Modern Enterprise Service Grids, with their brokered peer-to-peer architecture offer the next generation of efficiencies, ease of use, and savings for a wide array of distributed computing projects including the SOA market as summarized in Table 1. ■

About the Author

Atul Saini is CEO and CTO of Fiorano Software Inc. Under his leadership, Fiorano has emerged as a recognized leader in the standards-based integration middleware business.

atul@fiorano.com



WRITTEN BY KYLE GABHART

✚ Few topics evoke more groans and eye rolling from software engineers and Web developers than the dreaded “TESTING.”

Testing falls into the same category as documentation, refactoring code, dusting, and visiting the dentist. Put it off until the last minute, do as little as possible, do it quickly, and move on to something else. I must confess that I have the same visceral reaction to the thought of ‘testing’ as others do. Consequently, I approached the prospect of reviewing a testing tool with the loathing of visiting the dentist. I was very relieved to discover that Parasoft’s SOAtest 5.0 took a lot of the pain, frustration, and busy work out of the testing experience.

Parasoft’s SOAtest 5.0 is a comprehensive testing and analysis tool suite tailored to the unique testing and validation needs of Service Oriented Architectures. It supports functional testing, scenario-based testing, stress testing, client testing with a mock service, and a whole range of validation capabilities (XML Schema, WSDL, WS-Security, BPML, etc.). SOAtest 5.0 then further supports

the creation and automated execution of regression test suites. It supports a broad range of SOA specifications and standards, and is designed from the ground up to support the dynamic and evolving nature of service-oriented systems.

Getting Started with SOAtest 5.0

Installing SOAtest was a breeze. The process is as follows:

- Run the setup executable from media or by downloading it from www.parasoft.com,
- Follow the on-screen prompts.
- Decide if you want to set up SOAtest as a Windows service.

Once the software is installed, you’re ready to begin using SOAtest. The first time that you launch the software, you’ll be prompted either to input your individual license information or point to an available license server.

After providing licensing information, you will be confronted with an option to create a New Project, Open an Existing Project, or access the SOAtest Tutorial. The tutorial is well-written and provides a nice tour of the tool’s major features.

Creating Test Cases

Tests can either be created individually or as a part of a larger test suite. The tool seems to drive you toward creating test suites rather than individual tests, which is nice for reuse, organization, and best of all – regression testing a collection of tests. SOAtest supports the creation of test suites from a wide range of sources including:

- Web Services Description Language (WSDL) files
- Business Process Execution Language (BPEL) files
- Universal Description, Discovery & Integration (UDDI) registry end-points
- Web Services Inspection Language (WSIL) files
- BEA Aqualogic Enterprise Repository end-points

The test suite creation wizard will ask you to designate a file, URL, or end-point to query and also ask you to designate what tests you want to create. Upon completing the wizard, you'll have a whole set of tests automatically generated and ready-to-run as is or customized prior to execution.

Functional Testing

I was very impressed with the functional verification testing capabilities. SOAtest supports the following functional verification features:

- Check schema validity against a WSDL
- XML-aware diff engine that flags only true XML structure changes
- Surgical inclusion/exclusion of XML message elements in the verification process via XPATH (see Figure 1)
- A graphical rules engine for managing assertions

Every one of these features is enabled through simple intuitive menu options, dropdown lists, check boxes, and XML tree structures. The interface is simple to navigate and although there's a wealth of options, they're organized to avoid the feeling of being overwhelmed.

Executing one or more functional tests multiple times based on an external data set (database or spreadsheet) was another feature that I sought out. SOAtest came through in this area as well supporting the ability to point at a data source (CSV, Excel, relational databases, etc.) and run through a battery of tests that pull values out of the data source and send service requests containing the extracted data values. This lets you define a single test case or suite of tests and then automatically test the full range of data values that the test case needs to support without creating additional tests for each value.

Testing with Mock Services

In a truly test-driven environment, I'd create service interfaces and corresponding suites of test cases before any implementation code is developed. Although this is good in theory, without a supporting toolset, it's not very realistic. SOAtest provides all the capabilities needed to support this kind of development model. When creating a new project/test suite, simply point the tool at a WSDL file and check the "Generate Server Stubs" radio button. The server stubs will run on SOAtest's embedded Tomcat server and let you create and run testing scenarios before you've written a single line of implementation code. This way, when you're ready to implement the service interfaces, you have a defined a suite of tests for verification. This helps to control the scope (when you meet all the tests, stop working) and provides one or more test suites that can be incorporated into automatic regression testing (see below) to ensure that functionality isn't compromised as the project progresses.

Of all the features that I've worked with, I found the service mock-up process the most cumbersome. The other aspects of SOAtest were intuitive and easy-to-work with, whereas I had to wrestle a while to get the mock service capability working on

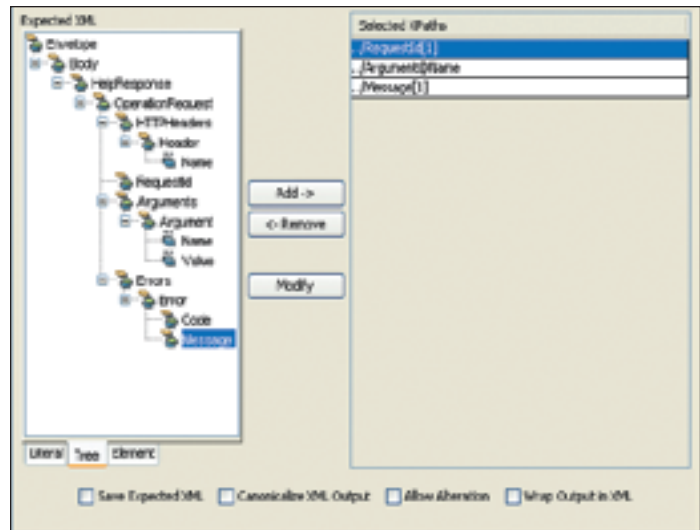


Figure 1: You can use XPATH to surgically select what portions of an XML message you want to include in the validation.



Figure 2: Testing results can be displayed graphically through summary and detailed reports

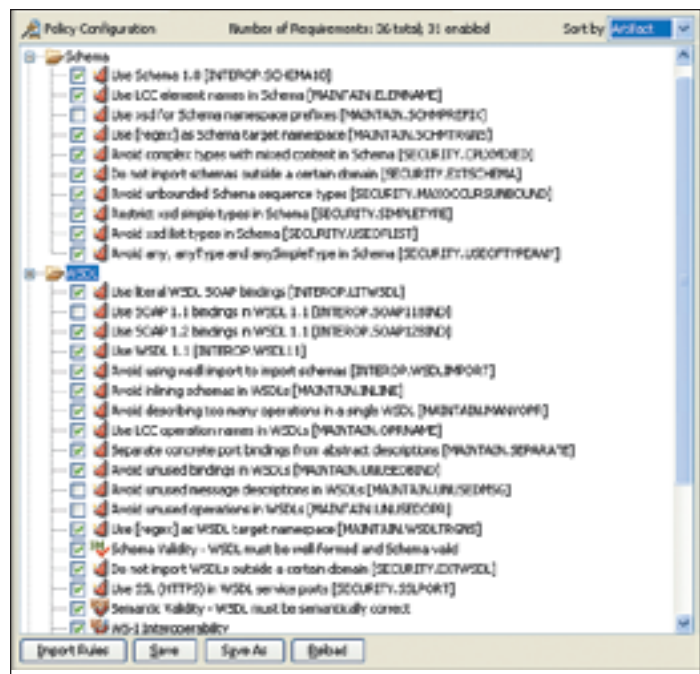


Figure 3: SOAtest provides a simple GUI for configuring service design and development policies

anything other than the tutorial walk-through. In the end, the functionality of this capability was excellent, albeit a bit difficult to initially configure.

Regression Testing

Running your tests once is nice. Running your tests regressively is better. Running your tests regressively and automatically at night is divine. SOAtest supports all three scenarios. Any of the test suites that you define in a SOAtest project can be converted to regression test suites. Furthermore, using XPATH you can selectively indicate which portions of the test cases may change from test to test and which values should never change. Once you have a set of regression tests that you're happy with, SOAtest provides a command-line mechanism to kick off your test suites automatically. Thus in an agile, continuous integration environment you can run regression testing at night, at lunch, or every hour on the hour to ensure that you find bugs early and often.

Reporting

SOAtest provides several reporting features (auto-generated reports for nightly regression tests, on-demand, detailed summary reports for test suites, and WS-I interoperability reports). I found the quality and readability of the reports developed by SOAtest to be quite good. Figure 2 provides a snapshot of part of the SOAtest detailed report. The WS-I interoperability reports were pretty low quality, difficult to navigate, and provided information overload. In fairness to SOAtest, those reports are copyrighted by WS-I and seem to be auto-generated by one or more WS-I tools. Consequently, I'm not sure that Parasoft has any control over the quality of these reports. Nonetheless, I would have liked a WS-I conformance report of the same quality and user-friendliness as the native SOAtest reports.

Advanced Testing Features

The palette of testing and analysis features in SOAtest is extensive. In the interest of not filling up this entire magazine with

feature descriptions, I'll list several of the compelling features that I've not covered already:

- Scenario-based testing where subsequent test cases depend on data returned from previous test cases
- Stress/load testing your SOA and specifying Quality of Service parameters
- Testing non-XML services (JMS, MQ, TIBCO, EJB, REST, Binary, Text, etc.)
- Validate SOAP security using WS-Security (encryption, digital signatures, and authentication)
- Asynchronous service testing
- Custom scripting with Python, JavaScript, or Java

Drawbacks

As I mentioned earlier, the mock service feature is a bit awkward to work with initially. Also, the WS-I conformance report was not up to the same quality standards as the native SOAtest reports. With SOA's strong integration and interoperability play, there are a lot of enterprises that have Java and other services that all need to be involved in the same business process and testing scenario.

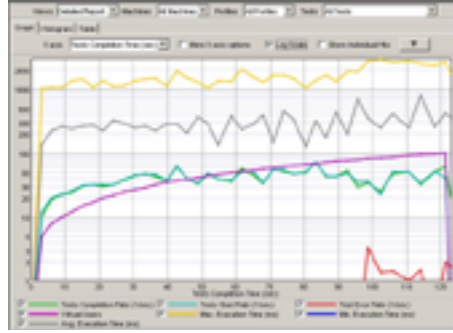
Summary

In spite of my general dislike for software analysis and testing, I found Parasoft's SOAtest 5.0 to be a well-designed tool that took a lot of the pain and work out of testing and validating a SOA. The tool isn't perfect, but it is easily one of the best SOA testing tools that I've ever worked with. ■

About the Author

Kyle Gabhart is a subject matter expert specializing in Service Oriented technologies who currently serves as the SOA Lead for Web Age Solutions, a premier provider of technology education and mentoring. Since 2001 he has contributed extensively to the SOA community as an author, speaker, consultant, and open source contributor.

kgabhart@webagesolutions.com



Product Snapshot

Target Audience: SOA architects, developers, QA/testers, and analysts

Level: Beginner to Advanced

Pros: Powerful, intuitive UI, robust testing and analysis tools

Cons: The mock service feature is a bit awkward, only supports Java clients for now

Testing environment: Dell Inspiron 640m, 1.6GHz Intel Core-Duo, 2GB RAM, Windows XP Pro with SP2

Platforms: Windows 2000/XP, Linux, Solaris

IBM®





WebSphere®

`_INFRASTRUCTURE LOG`

`_DAY 18: Came to work and found everything frozen. Icicles are everywhere. It's our processes. They're inflexible. Hard coded so we can't respond to change.`

`_Why did we lock ourselves in like this? Brrrr.`

`_DAY 19: A way out. IBM WebSphere middleware for Business Process Management. It lets us streamline business tasks and optimize performance. We can simulate and test our processes so we understand the impact they'll have, then monitor performance once they're deployed. And because it's based on a service oriented architecture, it's easy to reuse and connect existing process-based services.`

`_Everything's unfrozen now. Wow, it's good to feel my toes again.`

Take the BPM with SOA Assessment at:
IBM.COM/[TAKEBACKCONTROL/PROCESS](http://IBM.COM/TAKEBACKCONTROL/PROCESS)

Security in a SOA

Key to achieving the plug-and-play goals of SOA



WRITTEN BY RAJIV GUPTA

➤ As the name suggests, a Service Oriented Architecture is one where application functionality is packaged as autonomous services that adhere to industry standard interfaces (WSDL, SOAP), and the services are then deployed in an IT architecture that makes for their most effective use.

The component services can be rapidly reused and composited, plugged and played as it were, to create new business offerings and they can be individually upgraded for increased business agility. However, to achieve the promise of a SOA it's imperative that critical non-business logic-related functionality, the foremost of which is security, should also be provided and used as a service. And for this to occur it has to be externalized, accessed, and managed independently from the business logic-related services.

This article will address application security, specifically fine-grained application access control or entitlements. The need to externalize and independently manage security is many:

1. The security context in which a component service will be executed will be a function of the composite service in which it will be invoked. It can't be determined when the code for the component service is developed. If security that pertains to access, or who can use the service, is codified in the component service then this code will have to be modified to use the component service in different environments or in different composite services, defeating the plug-and-play benefits that are a key driver for a SOA.
2. The owners, administrators, and specialists of an access policy are different from those developing the business logic of the component service. They are often in different organizational

domains. Requiring that the access policy management be coded at the same time and in the same package as the business logic requires coordination that is unnecessary and inefficient.

3. Access policies need to be audited and checked outside of the service for purposes of compliance. Access policy auditing is required for the component service as well as for the composite service or business process in which the component service is invoked. Auditing requirements are important for corporate governance and compliance, but perhaps more importantly because in the pre-SOA world there were only a few, very limited, very controlled ways in which the application functionality could be used. However, in a SOA world the service can be invoked in very diverse and unanticipated ways. Auditing is a key tool to anticipate problems before they occur and locate the root cause of problems when they occur.

In a well-designed SOA, access policy management is itself an important service, sometimes referred to as an infrastructure service. This is in contrast to the component services and composite services that encode the business logic, sometimes referred to as business services.

So what does externalizing access policy management from the component service really mean? The figure below depicts pre-SOA application functionality where security is managed in the business logic, moving to a SOA-compatible service version of the same business logic with access policy administration, resolution, and auditing externalized from the service and manifest as SOA-compatible infrastructure services themselves. In some cases the enforcement of the access policy will also be externalized from the service. Generally this won't be possible and the service can get the access policy decision from the external security service and enforce the decision itself.

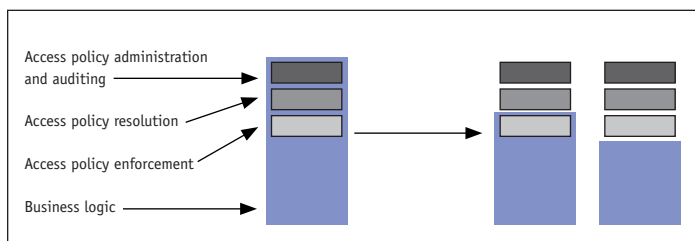


Figure 1

It's less critical that access policy enforcement be manifest as a separate infrastructure service because it's often tied very closely to the business logic and changes with the business logic, unlike access policy administration, auditing, or resolution which change at a different rate, at different times, and are owned by people other than the developers of the business logic.

As an example consider the following:

- **Component service:** Order management.
- **Access policy:** An order can be entered only by people with the role of "broker." An order can be updated only by the owner of the order or by a manager of the owner of the order. An order can be read by the owner of the order, a manager of the owner of the order, or by a subject with the role "reconcile."

In this case the developers of the order management service can focus purely on implementing the most efficient order management functionality. The access policy has to specify who can access the service and perform the function exposed by the order management service. The policy needs to be administered outside of the order management service, potentially by multiple independent people (more on that later). The access policy resolution needs to access the appropriate contextual information, for example, accessing a central LDAP in which the roles of the users may be stored, or another in which the user-manager relationship may be stored, or a local database in which the order-owner relationship may be stored. The policy resolution will determine if the given request should be permitted or denied, and the policy enforcement will enforce that decision on the request.

The benefit of separating the access policy administration, resolution, and audit from the component service is that the access policy can be changed to comply with changing security or compliance requirements without requiring any change or recoding of the component service. For example, a Patriot Act rule might require that an order from a user with a certain attribute, say if he belongs to a set of identified organizations, with a value over a certain amount shouldn't be permitted. Or a Sarbanes-Oxley rule might require that the duties between a broker and an administrator be segregated. Or a business situation such as M&A may require that the access policy be modified to permit access to users who have the role of broker in one system or account manager in another system. These changes, which are independent of the business logic of the order management service can be effected by simply configuring a new or modified access policy and without requiring a change in the order management service itself.

Conversely, a new more efficient order management service won't require a recertification of the existing access policies. As the SOA deployment in an organization matures, the component services will be invoked in composite services that will determine part of the security context in which the invocation of the component service will have to be checked. In the figure below the order management component service may be invoked in a "reconcile all day orders" composite service or in a new "compute commission" composite service.

The access policy can take into account the context of the composite service from which the component service is being invoked, who is initiating the composite service, etc. The order management component service can now be used in ways unanticipated at the time the service was developed, and the appropriate access policy can be administered, resolved, and audited without losing security or compliance and without requiring any reworking of the component service.

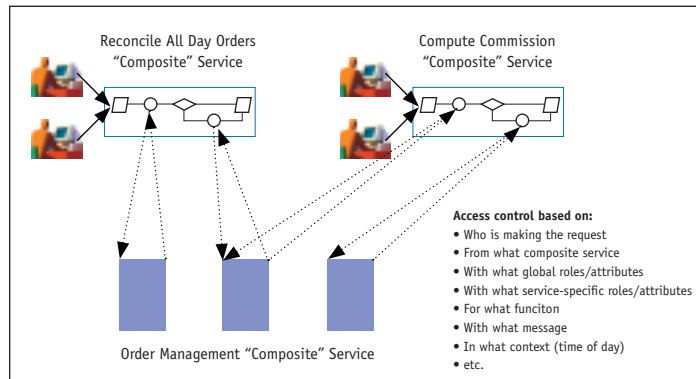


Figure 2

So where does one begin?

Begin by making sure that access policy management – administration, resolution, auditing – aren't embedded inside a component or composite service. If the application logic is developed in a standard container model, e.g., J2EE or .NET, then try to make the service resources that need to be protected be at the granularity of the container interfaces. If so, then the access policy enforcement can be performed by a standards-based interceptor that can be integrated into the application infrastructure stack without any change in the application code. This interceptor-based enforcer will permit or deny a service resource to be accessed by permitting or denying the corresponding container interface from being invoked. Similarly, if the application logic has a standard invocation model, such as SOAP, and the granularity of the resources being protected are at the granularity of the invocation interface then the access policy enforcement can be performed by a standards-based interceptor in the SOAP stack. The interceptors can be deployed as code that is co-resident with the business service or they can be a separate infrastructure service that is invoked from within the application code. Generally the application code will invoke the policy resolution service over standard interfaces, e.g., XACML over SOAP, to get the access policy decision and will enforce the decision in the application code.

A well-designed SOA that is truly loosely coupled with access policy administration, resolution, and auditing as standards-compliant infrastructure services is depicted in the figure below.

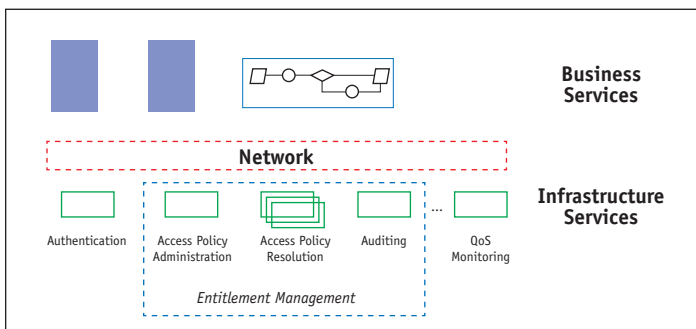


Figure 3

Unlike most other infrastructure services the Access Policy Resolution for fine-grained accesses or entitlements has constraints that dictate which particular instance of the Resolution service should be used. Since the access policy will be applied on every access and since the policy resolution may require message context and other attribute information that is local to the business service, it's likely that a relatively local instance of the resolution service will need to be invoked. It may be impractical from a performance, scalability, and availability point-of-view to use a centralized resolution service. Therefore it's important that a practical and effective security infrastructure for a SOA permit distributed Access Policy Resolution through multiple distributed instances of the Resolution service.

While this is the end state deployment architecture, how does one get there from here? What are some of the key operations and policy administration issues, and how does one deal with them?

Since there are many different owners of access policy for a given resource, e.g., component service administrator, composite service administrator, enterprise security and InfoSec teams, enterprise and LoB compliance teams, it's imperative that the policy administration service has a rich and effective delegation capability. It's critically important that the administration service not require all of these owners of access policy to coordinate their efforts or to have to come together and administer a single unified policy at the same time. Some of the conditions of the policy may need to be defined at different times. For example, the administrator of the composite service may want to have a say in the access policy of the component service at a much later time than the administrator of the component service will want to administer his say into the access policy of the component service. Similarly the compliance team will want to change the compliance aspects of the access policy, say an order initiator can't be the order approver, autonomously from the administration of the other aspects of the policy. In fact the compliance team will want the capability to change the policy to respond to a change in regulations without having to coordinate with the other administrators of access policy for a resource. In many instances it will be important from a checks-and-balances perspective that the administrators be different and independent.

Depending on the role a person is playing, when the person logs into the administration service they should only be able to administer those aspects of access policy for a resource that they are permitted to administer. So the administration service itself needs to be entitled, and needs to have rich delegation capability.

Now if there are to be many autonomous administrators of policy and if they aren't to be required to coordinate with each other, then it's possible that the policies that they define will be in conflict with each other. For example the administrator of the reconcile day orders composite service may specify a policy allowing access to the order management component service but the administrator of the order management component service may specify a policy that denies the access perhaps because the user on whose behalf the composite service is being initiated is also the approver of an order and this violates a segregation of duty policy at the order management component service. Therefore the administration infrastructure service should anticipate and handle access policy conflicts. These conflicts should be resolved when the access using the most up-to-date dynamic information and the resulting policy decision are enforced on the resource access.

A related and very important administration issue relates to user roles. Role-based access control (RBAC) or the use of roles

in access policies is a very important way to manage accesses to resources. The benefits of RBAC are many and have been well documented in many excellent articles. The key driver for RBAC is ease of management – there are typically many fewer roles than there are users. Since the same user can be a member of multiple projects, each project can have its own access requirements, and since user-to-project and project-to-access mappings can change, roles are a powerful abstraction to manage and enable this flexibility.

Important as RBAC may be, when deploying a security infrastructure service in a SOA (the security infrastructure service is more accurately a set of services – administration, resolution, and auditing – as mentioned earlier) role assignment can also be cause for paralysis. Whether deploying a SOA or not, many organizations first try to reconcile all roles across the enterprise in a top-down fashion. This is a long, painful, and largely futile exercise. There are a few enterprise-wide roles; most roles are resource-specific. Yes, you may be a “VP” in the corporate LDAP, but that by itself isn't going to let you access the development version of a business service. And that by itself shouldn't let you administer the HR service.

Each resource has its own notion of roles and what level of access should be allowed users with what roles. These resource-specific roles in conjunction with global roles can be the basis of an effective RBAC solution. For example, an access policy may state that access to a business service is allowed to users who have a “controller” role in the enterprise LDAP or an “administrator” role in the service being protected. The access policies should allow global and service-specific user roles to be specified and used (user attributes in general, where a role is one form of attribute; others could be geography, organizational membership, employment status, etc.). The security administration service should also allow resource owners to specify, assign, and manage resource-specific user attributes.

Such distributed ownership and management of resource-specific user attributes is consistent with an unstated principle that underlines SOA: local control, global coordination. It's not only necessary for the smooth functioning of a practical SOA, it also expedites getting to a state of meaningful and effective RBAC. Now instead of trying to reconcile all roles across the enterprise in a top-down fashion and trying to keep them all consistent when user-to-role or role-to-access mappings change, most role assignments are delegated to the resource owners who can define what they need for their particular resource and can administer and manage appropriate changes at an appropriate pace.

In conclusion, a Service Oriented Architecture is more than simply packaging application functionality into business services that adhere to industry standard interfaces. It requires the externalization of non-business logic-related functionality from the applications that have to be provided and used as a set of standards-compliant infrastructure services. Security is a critical infrastructure service that is key to achieving the plug-and-play goals of SOA. If designed well it can abet the smooth operation and evolution of a SOA environment, and more importantly it can smooth the path to a SOA environment. If not, it can be the undoing of an otherwise sound SOA plan. ■

About the Author

Rajiv Gupta is CEO and founder of Securent, Inc., a leading provider of entitlement management solutions.

rgupta@securent.net

“Businesses that ignore the potential of SOA will find themselves outpaced by rivals who improve their agility and transform themselves into new kinds of enterprises

— Yafim Natis, Gartner Analyst

3-DAY EVENT!

SOAWorld

Plus **2007**

Enterprise OpenSource Conference & Expo 2007

TOPICS INCLUDE:

SOA Web Services

- > AJAX and SOA
- > Web 2.0
- > Universal SOA
- > Protecting Web Services
- > Troubleshooting SOA
- > Governance
- > Open-Source SOA
- > XBRL
- > Service Virtualization

Open Source

- > Open Source Business Models
- > Open Source ESB
- > OpenAjax Alliance
- > SaaS and Open Source
- > Spring, Hibernate and Eclipse
- > Seam
- > Open Source Penetration
- > Monetizing Open Source
- > Open Source Databases
- > AMQP
- > Open Source Middleware

June 25-27, 2007

Roosevelt Hotel / New York City

Register Online! www.SOAWorld2007.com

11th International
SOAWorldTM
CONFERENCE & EXPO

2007 is to many industry insiders shaping up to be a major inflection point in software development and deployment, with SOA, Web Services, Open Source, and AJAX all converging as cross-platform and cross-browser apps become the rule rather than the exception.

Accordingly the 11th International SOA Web Services Edge 2007 again seeks to offer comprehensive coverage and actionable insights to the developers, architects, IT managers, CXOs, analysts, VCs, and journalists who'll be assembling as delegates and VIP guests in The Roosevelt Hotel in downtown Manhattan, June 25-27, 2007

Co-located with the 2nd Annual Enterprise Open Source Conference & Expo, the event will deliver the #1 i-technology educational and networking opportunity of the year. These two conference programs between them will present a comprehensive view of all the development and management aspects of integrating a SOA strategy and an Open Source philosophy into your enterprise. Our organizing principle is that delegates will go away from the intense two-day program replete with why-to and how-to knowledge delivered first-hand by industry experts.

Visit soaeosconference.sys-con.com for the most up-to-the-minute information including... Keynotes, Sessions, Speakers, Sponsors, Exhibitors, Schedule, etc.

2nd Annual
ENTERPRISE > 2007
OPENSOURCE
CONFERENCE+EXPO

SOAEOSCONFERENCE.SYS-CON.COM

REGISTER ONLINE TODAY

SAVE \$200!

(HURRY FOR EARLY-BIRD DISCOUNT)

BROUGHT TO YOU BY:



» **SOA World Magazine**
focuses on the business and technology of Service-Oriented Architectures and Web Services. It targets enterprise application development and management, in all its aspects.



» **Enterprise Open Source Magazine**
EOS is the world's leading publication showcasing every aspect of profitable Open Source solutions in business and consumer contexts.



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

Exhibit and Sponsorship Info:

Call 201-802-3020 or email events@sys-con.com



WRITTEN BY MICHAEL HAVEY

➤ The natural visualization of a business process is of boxes and arrows arranged in a tree-like formation. A large process with numerous conditional paths forms a rather expansive tree that can't fit on a computer screen or printed page. If the process has loops, these are often represented as arrows pointing back to earlier boxes, resulting in an untidy *graph* structure. Although BPEL isn't a visual process language, its XML representation can form *code trees* that are no less cumbersome. A receive inside a sequence inside a flow inside a switch inside a pick, even if properly indented, can make a coder see double.

This technique article shows how to model BPEL 1.1 processes in a special flat form that represents even the most onerous processes in just a few levels of structure. A process modeled in this form, represented visually, more closely resembles a neat pile of sticks than a tree. Aesthetics aside, the flat approach is fundamentally better suited to SOA orchestration than the tree approach. Flat BPEL is good SOA.

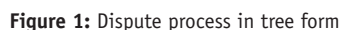
Flat Credit Card Dispute

To demonstrate the flat technique, let's consider the (deliberately complicated) example of how a bank processes credit card disputes raised by its customers. The following is an English description of ACMEBank's business process for disputes on its personal Visa cards:

- **The Capture Stage:** The customer submits a dispute over one of various channels (call center, Web, e-mail, mail, or fax) to ACMEBank. The bank makes a quick determination whether to reject the dispute outright (e.g., because it wasn't submitted within 60 days of its posted date), write it off (e.g., because the dollar amount of the transaction is below a certain threshold), request from the customer supporting documentation (e.g., receipts), or pass it to one of the bank's dispute specialists for further investigation. At any time at this stage the customer may cancel the dispute. The goal of this stage is either to capture the dispute completely or to dispense with it.
- **The Investigation Stage:** The dispute specialist examines the fully captured dispute, and may reject it, write it off, proceed with a chargeback (i.e., take the disputed amount back from the merchant), or request a sales draft from the merchant to investigate its validity (and ask the customer for clarification if the draft appears valid). The happy path culminates in a chargeback decision.
- **The Chargeback Stage:** ACMEBank charges back the transaction and credits the customer's account. The merchant bank may, in turn, accept the chargeback or *represent* the transaction (i.e., present the charge again), at which point an ACMEBank dispute specialist determines whether to charge back a second time, write off, or reject. In rare cases, the dispute is brought to arbitration before the Visa association.

The process is designed in the style of a procedural program. The flow is deep and meandering. Activities are lost in the machinery of control structures. The most important steps in a BPEL process are its partner interactions (receives, onMessage handlers and invokes), but in this process they are scattered about the flow. Consequently, it's hard to determine how faithfully this process honors its contract with its partners; it's hard to see the *orchestration* in

The tree version of the disputes process is no straw man. Most business processes today are constructed in this fashion, largely because process designers lack process design sophistication. Business analysts, not surprisingly, understand the business problem domain but don't draw rigorous flowcharts. Technical designers, who are handed business analyst process diagrams in the transition from the requirements to the development stage, work mainly at hardening the process and plugging holes. They don't reshape or introduce patterns; processes remain procedural. Technical design perpetuates the bad practices of business analysis. Ironically, many



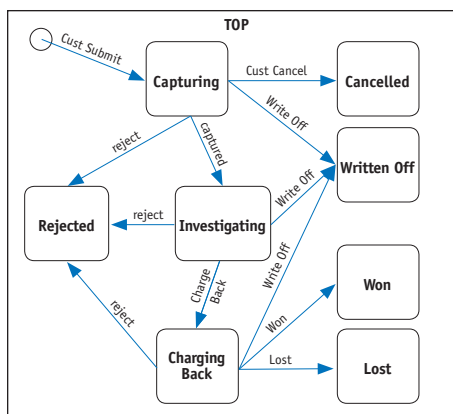


Figure 2: Top-level state diagram for dispute

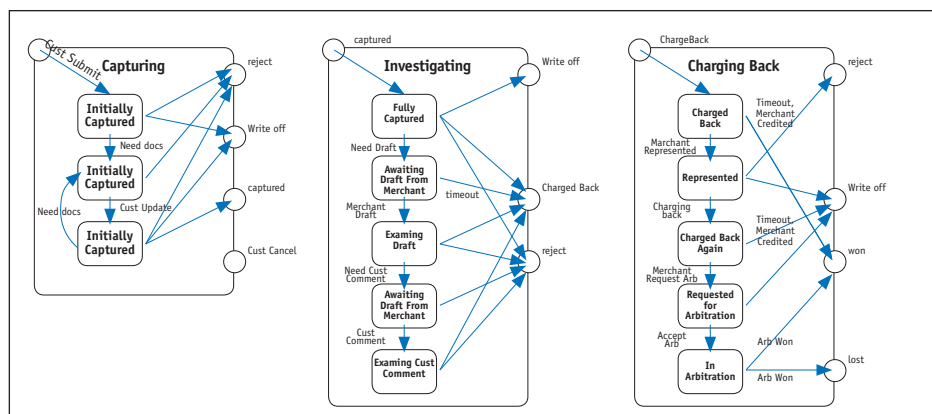


Figure 3: Sub-states for dispute

of these designers write beautiful state-of-the-art *services*: their Java or C# is impeccable, but they aren't strong at designing *processes* around these services.

The flat style of process design recommended in this article kills two birds with one stone. On the one hand, it encourages flat processes organized around partner interactions rather than procedural control structures. On the other hand, the emphasis on partner interactions is a purer expression of SOA orchestration than the alternative tree approach.

The trick is to conceive of the process as the *state machine* of some entity, observed from the point of view of the participant that manages the process — e.g., a credit card dispute for ACMEBank, a claim for an insurance company, a purchase order for a supplier. Transitions between states are triggered by inbound events from other partners: receives, picks, and handlers. The process performs actions either during a transition or when a state is entered or exited. The most significant actions from an SOA perspective are invokes in which the current participant calls other participants, possibly triggering state changes in them too. Once the states, transitions, and actions of the state machine are understood, the BPEL process flow is easy to develop.

The state machine of an ACME dispute, drawn as a state diagram, is shown in Figure 2 and Figure 3. The first figure depicts the top-level view. The three stages of the disputes process are modeled as three states: Capturing, Investigating, Charging Back. Capturing is the initial state, triggered by the event Cust Submit. Rejected, Written Off, Cancelled, Won, and Lost are final states in which the processing of the dispute is complete. Capturing, Investigating, and Charging Back have sub-states shown in Figure 3. The Capturing state, for example, begins in the Initially Captured state, and moves to the Awaiting Docs state on the Need Docs transition. Awaiting Docs, in turn, transitions to Examining Docs on Cust Update, and moves back to Awaiting Docs if more docs are needed (Need Docs). The dispute can be rejected, written off, or declared captured in either the Initially Captured or Examining Docs states. In any of the states, the customer may cancel (Cust Cancel).

Figure 4 is a visual representation of the flat BPEL process. The process begins with an initial event (receive CustChannel Submitted) that starts up the machine. The main logic is a switch within a while. Each iteration of the while handles one state: it enters the state, performs any entry actions, and waits for the next event to transition to the next state. The while runs for as long as the machine is active; when a final state is entered, it sets the Boolean

variable Continue to false to stop the loop. The switch inside the while has cases for each top-level state (for readability, these boxes are shaded and have a thick border); the current state is identified by the process variable state, whose initial value is Capturing. There are five top-level cases (Cancelled, Capturing, Investigating, and Charging Back have one case each, and Rejected, Written Off, Won, and Lost share the fifth). The three hierarchical states — Capturing, Investigating, and Charging Back — each have an internal switch with a case for each sub-state; they use the variables capturingState, investigatingState, and chargingBackState respectively to select the appropriate case. Transitioning out of a given state is modeled with a pick, which receives one of multiple events, performs actions required to handle that event, and sets a state variable to advance the state. That state variable is either state (which advances to another top-level state), or capturingState, investigatingState, or chargingBackState (to advance to a different sub-state within the same top-level state).

To see the mechanics of this, consider Listing 1, an excerpt of the actual BPEL disputes process. (The complete code can be downloaded from <http://webservices.sys-con.com/read/issue/archives/>.) If the current top-level state is Investigating and its current sub-state is FullyCaptured, the process first asks operations to pick up the dispute for investigation (invoke on line 54), and then uses a pick to wait for operations' response. If the response is to reject (onMessage in line 56), the process sets the top-level state to Rejected (assign in lines 57-62), and on the next iteration will jump to the top-level case for Rejected (line 33). If the response is to request documentation from the merchant (onMessage on line 80), the process keeps the top-level state as is, but sets the sub-state to AwaitingDraft (assign in lines 81-86); on the next iteration, the process will jump to the AwaitingDraftcase (line 92).

Although there are several levels of hierarchy in the flat approach (transitions on a sub-state, for example, are modeled in a pick within a switch within a switch within a while), there is an upper bound on the number of levels. State machines, even hierarchical ones like the dispute machine, are comparatively flat; even the most complex actors seldom require more than three levels of state. If companies paid for their processes by the level, meandering tree processes would exceed their budget quickly, whereas flat processes would incur a predictable charge up-front but stay on budget. Flat processes would also pass an SOA audit with flying colors, because in the flat structure states, transitions, and actions are easy to find.

JavaOne™

Sun's 2007 Worldwide Java Developer Conference™

May 8-11, 2007

The Moscone Center
San Francisco, CA

JavaOne™ Pavilion: May 8-10, 2007



IT'S AN EXCITING TIME FOR THE ENTIRE JAVA™ TECHNOLOGY COMMUNITY.

With the evolution of the Java platform, new opportunities are emerging for developers, thought leaders, and entrepreneurs. That's why for 2007 the Conference is featuring an expanded program that embraces technologies outside the core Java platform while keeping Java™ technology at the center. You'll get the same in-depth content we have always focused on, plus more topics and issues relevant to today's evolving market.

LEARN MORE ABOUT:*

- > Java Technology
- > Scripting
- > Open Source and Community Development
- > Integration and Services-Oriented Development
- > Web 2.0
- > Compatibility and Interoperability
- > Business Management
- > And More



SAVE \$200**
REGISTER BY
APRIL 4, 2007

Please use priority code: J7PA1SOA

*Content subject to change.
**Offer not available on site.

Register Today @ java.sun.com/javaone



Handling Concurrency and Other Conversion Tips

Table 1 provides some tips on how to map, element-by-element, existing BPEL processes to the flat form. The disputes example presented above demonstrates most of these tips.

Concurrency is a notable exception. In BPEL, concurrent execution of activities is modeled with a flow activity. There are two sorts of flows: those that merely perform a set of actions in parallel and those that model what state machine guru David Harel calls *orthogonal* states (or a set of states that apply to the same entity simultaneously). The former type is easily accommodated in the flat approach: the flow construct is simply embedded in the logic of a particular state or transition. The latter type contains *transitions* (receives, waits, or picks), and thus affects the shape of the state machine. In the latter case the flat approach of modeling state as a single switch within a while breaks down, because it assumes states are mutually exclusive. Two possible design approaches are to *flatten* the flow into mutually exclusive form, or to explicitly build the logic to support orthogonal states.

A good example is presented in the Voting process presented in the BPMN specification. In the “Collecting Votes” part of this process, three main activities occur in parallel: a moderated e-mail discussion, a moderated conference call, and the collection of votes. A visual representation of the BPEL implementation recommended in the spec is shown in Figure 5.

The model shown in the figure, which features a disconnected set of e-mail activities (Invoke Moderate Email Discussion, despite appearances, is closely related to the sequence Wait for 6 Days and Invoke Email Deadline Notice) and a time-based exception to break an infinite while loop, is obscure to say the least. The overall meaning is clearer when represented as a state machine with orthogonal states, as shown in Figure 6. According to the diagram, the voting procedure has, simultaneously, a conference call state, an e-mail state, and a voting state.

Figure 7 is a BPEL representation in a variation of the flat form in which the set of orthogonal states is a flow, and each of the orthogonal states is a while within the flow. Notice that each while has its own internal state-transition structure; each while is a little state machine, running in parallel with the state machines of peer states.

The flow, of course, is encompassed within the overall while-switch structure of the larger state machine, the same overall structure used in the disputes example.

BPEL Features	Usage in Flat
Receive, Pick, Wait, Handlers	Model as transition events.
Invoke, Reply	Keep as transition or state actions.
Sequence, Switch, While	Keep smaller instances if they work as state or transition actions. Break up larger instances as part of the overall transformation.
Assign and process variables.	Keep manipulations of business-meaningful data. Replace existing control flow data (e.g., flags to control switch or while) with state-related data.
Scope	Scoped handlers might signify “group” transitions.
Throw, Compensate	Treat as “group” transitions.
Flow	Treat with care, as discussed in the article.

Table 1: Mapping tree to flat

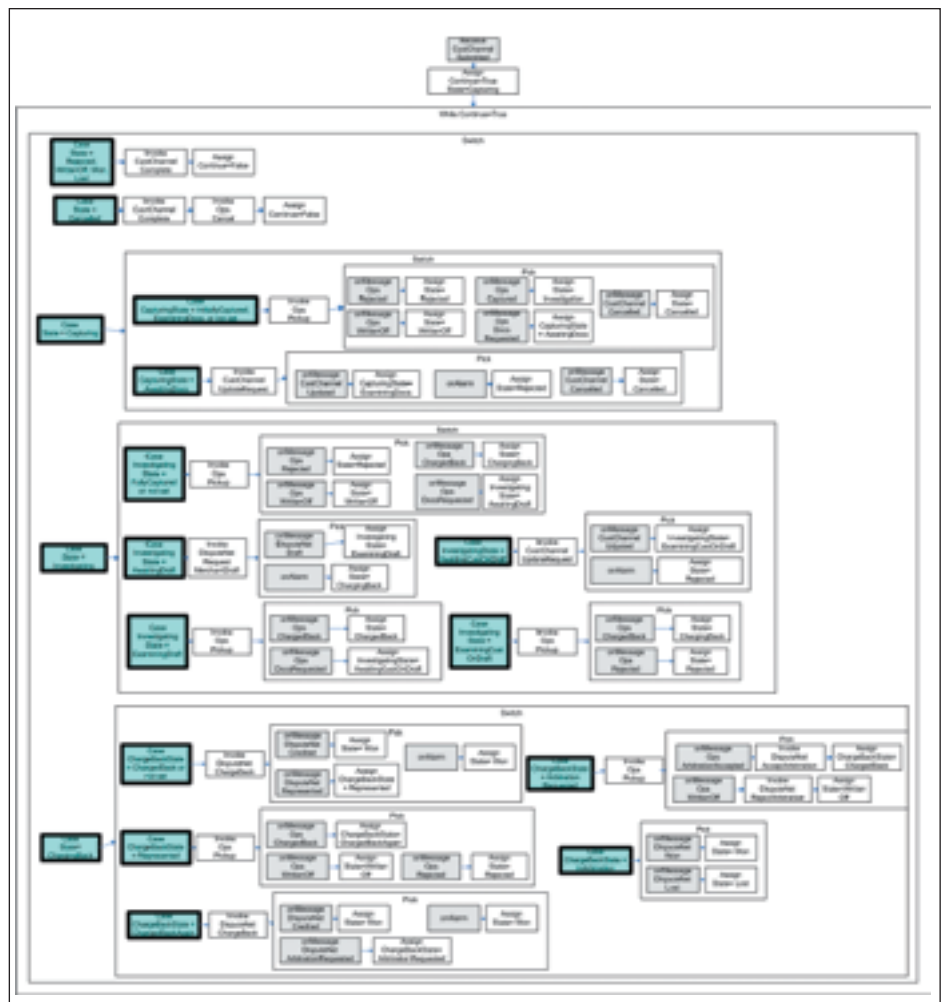


Figure 4: Dispute process in flat form

Although this representation has expressive power, it's complex — much more so, in fact, than the obscure but barren implementation from the BPMN spec! An alternative is to flatten the structure, reducing the voting cycle to three mutually exclusive states, as shown in Part (a) of Figure 8. In Collecting Votes Not Started all the setup work is done, which includes starting timers whose expiration events will lead to major transitions later on. Most of the time is spent in the state Collecting Votes, and of its four transitions, three are self-transi-

Listing 1 BPEL “Flat” Code Snippet

```

001 <process>
002 <variables>
003   <variable name="state" type="ns2:string"/>
004   <variable name="CapturingState" type="ns2:string"/>
005   <variable name="investigatingState" type="xsd:string"/>
006   <variable name="chargingBackState" type="xsd:string"/>
007   <variable name="loopContinue" type="ns2:boolean"/>
008   AND OTHERS
009 </variables>
010 <sequence>
011   <receive partnerLink="client" operations="submitted" />
012   <assign name="initVars">
013     <copy>
014       <from expression="string('Capturing')"/> <to variable="state"/>
015     </copy>
016     <copy>
017       <from expression="string('InitiallyCaptured')"/>
018       <to variable="capturingState"/>
019     </copy>
020     <copy>
021       <from expression="string('FullyCaptured')"/>
022       <to variable="investigatingState"/>
023     </copy>
024     <copy>
025       <from expression="string('ChargedBack')"/> <to variable="char
                                gingBackState"/>
026     </copy>
027     <copy>
028       <from expression="true()"/> <to variable="loopContinue"/>
029     </copy>
030   </assign>
031   <while condition="bpws:getVariableData('loopContinue')=true()">
032     <switch name="topStateSwitch">
033       <case condition="bpws:getVariableData('state')='Rejected' or
034         bpws:getVariableData('state')='WrittenOff' or
035         bpws:getVariableData('state')='Won' or
036         bpws:getVariableData('state')='Lost'">
037         <sequence>
038           <invoke partnerLink="client" operation="completed" />
039           <assign name="setDone">
040             <copy>
041               <from expression="false()"/> <to variable="loopContinue"/>
042             </copy>
043           </assign>
044         </sequence>
045       </case>
046       <case condition="bpws:getVariableData('state')='Capturing'">
047         OMITTED
048       </case>
049       <case condition="bpws:getVariableData('state')='Investigating'">
050         <switch>
051           <case condition=
052             "bpws:getVariableData('investigatingState')='FullyCaptured'">
053             <sequence>
054               <invoke partnerLink="ops" operation="pickup"/>
055             <pick>
056               <onMessage operation="rejected" partnerLink="ops">
057                 <assign name="setState">
058                   <copy>
059                     <from expression="string('Rejected')"/>
060                     <to variable="state"/>
061                   </copy>
062                 </assign>
063               </onMessage>
064               <onMessage operation="writtenOff" partnerLink="ops">
065                 <assign name="setState">
066                   <copy>
067                     <from expression="string('WrittenOff')"/>
068                     <to variable="state"/>
069                   </copy>
070                 </assign>
071               </onMessage>
072               <onMessage operation="chargedBack" partnerLink="ops">
073                 <assign name="setState">
074                   <copy>
075                     <from expression="string('ChargingBack')"/>
076                     <to variable="state"/>
077                   </copy>
078                 </assign>
079               </onMessage>
080               <onMessage operation="requestedDocs" partnerLink="ops">
081                 <assign name="setState">
082                   <copy>
083                     <from expression="string('AwaitingDraft')"/>
084                     <to variable="investigatingState"/>
085                   </copy>
086                 </assign>
087               </onMessage>
088             </pick>
089           </sequence>
090         </case>
091         <case condition=
092           "bpws:getVariableData('investigatingState')='AwaitingDraft'">
093           <sequence>
094             <invoke partnerLink="net" operation="requestDraft"/>
095             <pick>
096               <onMessage operation="draft" partnerLink="net">
097                 <assign name="setState">
098                   <copy>
099                     <from expression="string('ExaminingDraft')"/>
100                     <to variable="investigatingState"/>
101                   </copy>
102                 </assign>
103               </onMessage>
104               <onAlarm ...>
105                 <assign name="setState">
106                   <copy>
107                     <from expression="string('ChargingBack')"/>
108                     <to variable="state"/>
109                   </copy>
110                 </assign>
111               </onAlarm>
112             </pick>
113           </sequence>
114         </case>
115         <case condition=
116           "bpws:getVariableData('investigatingState')='ExaminingDraft'">
117           OMITTED
118         </case>
119         <case condition=
120           "bpws:getVariableData('investigatingState')='AwaitingCustOn
                                Draft'">
121           OMITTED
122         </case>
123         <case condition=
124           "bpws:getVariableData('investigatingState')='ExaminingCust
                                OnDraft'">
125           OMITTED
126         </case>
127         <otherwise>
128           <throw name="throwUp" faultName="bpws:selectionFailure"/>
129         </otherwise>
130       </switch>
131     </case>
132     <case condition="bpws:getVariableData('state')='ChargingBack'">
133       OMITTED
134     </case>
135     <case condition="bpws:getVariableData('state')='Cancelled'">
136       OMITTED
137     </case>
138     <otherwise>
139       <throw name="throwUp" faultName="bpws:selectionFailure"/>
140     </otherwise>
141   </switch>
142 </while>
143 </sequence>
144 </process>

```

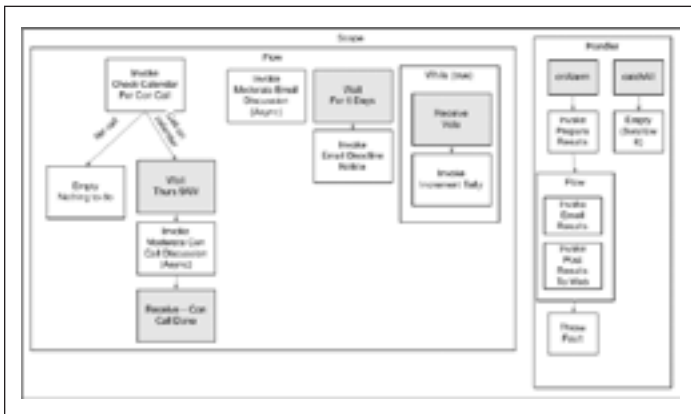


Figure 5: Collecting a votes BPEL process

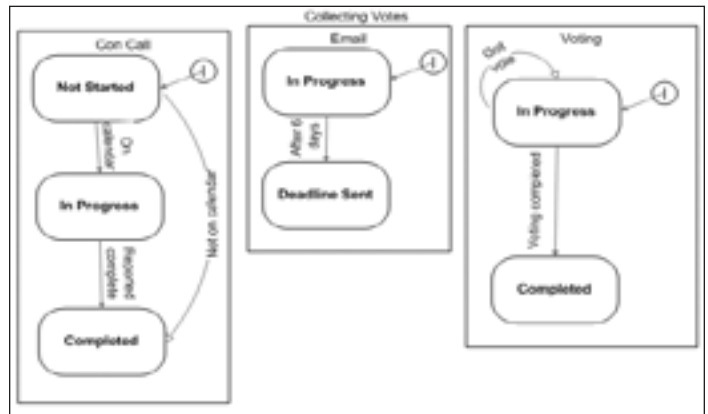


Figure 6: Collecting votes orthogonal state machine

tions whose occurrence keeps the machine in the Collecting Votes state. The machine moves to its final state, Collecting Votes Complete, when the voting period ends. The BPEL implementation, shown in Part (b), has the same overall while-switch form as the disputes example. The self-transitions in the middle state are a poor man's concurrency, preserving the requirement that events that impact the state machine can arrive concurrently, in unpredictable order.

Concurrency is a delicate subject, and the flattening heuristic does not work in all cases. The moral of the example is to preserve the spirit of flat even when faced with the challenge of orthogonal states. ■

References

1. Michael Havey. "State Machines and Workflow in Weblogic Integration: State Machines and Process-Oriented Applications." *Weblogic Developer's Journal*. January, 2004. <http://weblogic.sys-con.com/read/43046.htm>.
2. Michael Havey. *Essential Business Process Modeling*. O'Reilly. 2005. <http://www.oreilly.com/catalog/essentialpm>.
3. David Harel. "Statecharts: A Visual Formalism for Complex Systems." *Science of Computer Programming*, 8 (1987) 231-274. <http://www.tik.ee.ethz.ch/tik/education/lectures/hswcd/papers/Statecharts.pdf>.
4. Object Management Group. "Business Process Modeling Notation Specification." <http://www.bpmn.org/Documents/BPMN%20V1-0%20May%203%202004.pdf>.

About the Author

Michael Havey is a Chordiant consultant with a decade of industry experience, mostly with application integration. Michael's book *Essential Business Process Modeling* was published by O'Reilly in August 2005. michael.havey@chordiant.com

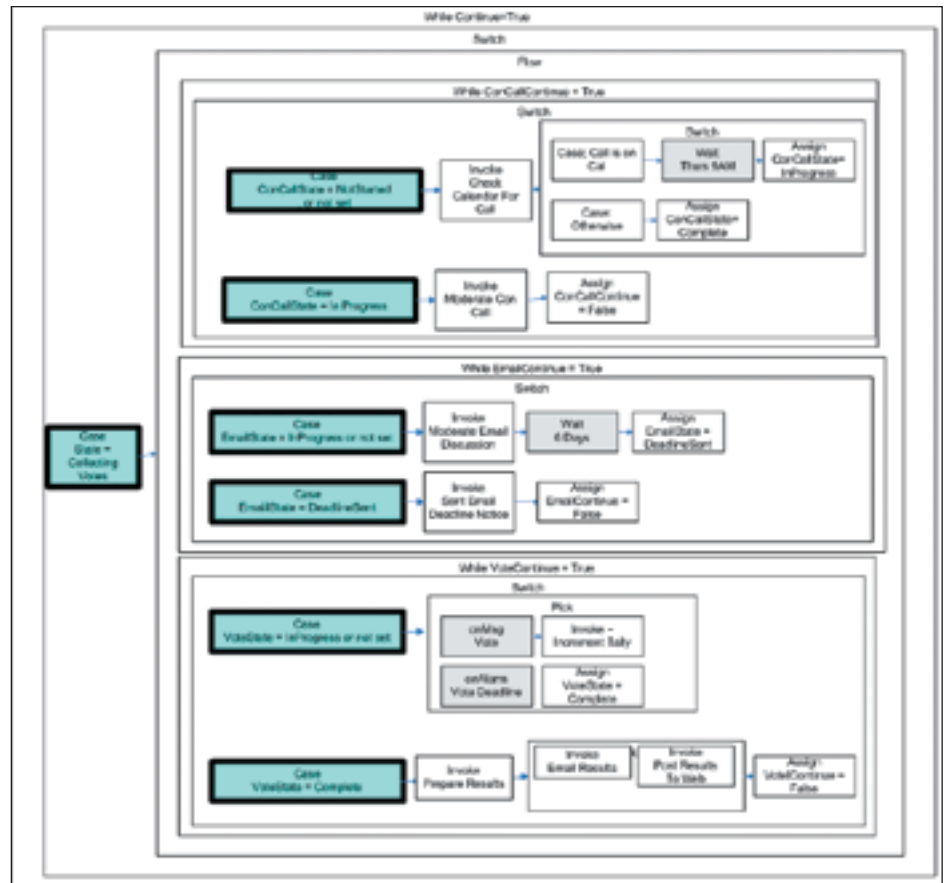


Figure 7: Collecting votes BPEL in orthogonal "flat" form

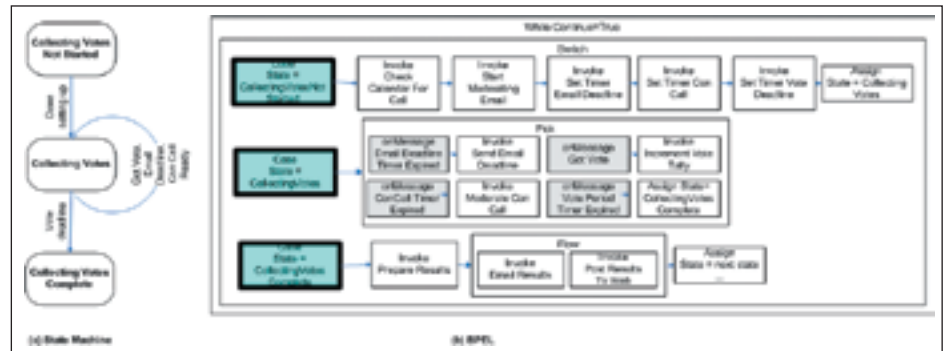


Figure 8: Collecting votes flattened

*The role of a **reference architecture** for housing would be to identify abstract solutions to the problems of providing housing. A general pattern for housing, one that addresses the needs of its occupants in the sense of, say, noting that there are bedrooms, kitchens, hallways, and so on is a good basis for an abstract reference architecture. The concept of eating area is a reference model concept, a kitchen is a realization of eating area in the context of the reference architecture.*

The goal of this reference model is to define the essence of service-oriented architecture and emerge with a vocabulary and a common understanding of SOA. It provides a normative reference that remains relevant for SOA, irrespective of the various and inevitable technology evolutions that will influence SOA deployment.

The concepts and relationships defined by the reference model are intended to be the basis for describing reference architectures and patterns that will define more specific categories of SOA designs. Concrete architectures arise from a combination of reference architectures, architectural patterns, and additional requirements, including those imposed by technology environments.


Again, abstraction, one is an instance(s) of the other, if I understand this correctly. I urge you to download and read this 31-page document to get some additional details. Also, get involved, if you like. I always love the comeback from standards bodies that deal with criticism and debate around the concepts they put forward: "If you don't like it, join us and change it." Can't argue with that.

I would, however, recommend that OASIS make this a bit easier to understand for the typical end user, as I stated before. Again, perhaps provide published use cases for the RM and RA, and thus make the notion a bit more consumable. I'll consider this an open-ended definition for now, so please send your use cases and I'll write columns about them here. In essence, how the reference architecture and model will be used in a sentence.

You have to give them an A for effort, but you can see where the debates are going to pop up. It's really a matter of understanding, a marketing issue really. I understand different levels of architectural abstractions, but what is really needed is a reference framework or model, and a set of steps to figure out how to build something that's proper for your problem domain. I've been building that recently, and I'm happy to map them back to this reference model if it makes sense for the project. The use cases out there are like snowflakes, and one size, or one model, does not fit all. You have to account for that within all this formality.

However, to be fair, if you Google "SOA Reference Model" or "SOA Model," or things of that nature, you'll find that most SOA vendors and consulting organizations have their own SOA (reference) models, SOA frameworks, SOA roadmaps, or SOA reference architectures. They are a bit different than what OASIS is proposing, but again, they're looking to solve the same sets of problems.

My best advice is to leverage the architecture, models, roadmaps, or frameworks that will work best for you. At the end of the day, I think clarity of approach will win over complexity, as long as you can do that without diminishing the value. ■




JOIN THE AJAX REVOLUTION!


Secrets of the Masters: Real-World AJAX

Edited by Dion Hinchcliffe & Kate Allen

"If you're looking for a one-stop shop for an AJAX book...you would have a long search to find a better overall resource than what you find in the chapters of this book."

Order Online at RealWorldAJAXBook.com and get
40% OFF Regular Bookstore Price!

 books.sys-con.com
from the World's Leading i-Technology Publisher



The Indispensable SOA

What it's doing to industries and how you can start with it

WRITTEN BY MICHAEL LIEBOW

➤ SOA has been aggressively hyped by the IT industry as a technology that can – and does – change the very nature of business. In recent times – as you well know – the Internet was a similar technology – and as with the Internet, we at IBM actually believe the hype to be true.

SOA has come on the scene in a big way in the past few years. But hype alone isn't what's causing the huge stir. It's because those who've started using SOA have discovered a huge benefit: flexibility. Let me put it this way: Before SOA, to have flexibility, a company might have needed to deploy and integrate 20 different software applications. After SOA, it mightn't need to build 20 isolated applications but only one instead – which, on a dime (both literally and figuratively), it can reconfigure 20 different ways to meet the imperatives of changing market conditions. That's flexibility.

Which begs the question: What's SOA? Without getting too technical, SOA is a technology that “decomposes” your software applications into their elemental parts – parts that you can rearrange, very fast, to create new applications. Thanks to open standards in both business and technology – standards that IBM along with the industry-at-large has relentlessly fought for in recent years – these components can be combined with those of your partners, customers, and suppliers. You can therefore create these super-applications, a composite of functionality, across companies and even industries, to develop entirely new kinds of business.

Suddenly, your IT is no longer an inhibitor to innovation. Fulfilling its promise to you, it's an enabler. And I can tell you that even now, early in its life, SOA is shaking up companies and industries.

We recently examined several leading-edge SOA projects covering 11 different industries that in one way or another exhibited

the recognizable pattern of consequences and benefits that the most acute observers predicted for SOA. Here, I'll talk about them briefly.

The network effect: Dominant suppliers and buyers, appreciating the simplicity and ease of SOA, can command their partners to follow suit. If these partners don't then they can lose these vital relationships. Consider what happened with radio frequency identification (RFID) a few years ago. An important buyer, convinced of its advantages, deployed it and compelled its network of suppliers to do so too. We have every reason to believe that SOA – with benefits at least as significant as those of RFID – will unfold the same way.

Intensified competition: Perhaps the most profound repercussions will occur in the competitive arena. SOA will likely intensify competition between existing competitors, and conceivably – out of seemingly nowhere – usher in new ones. SOA will do this by eliminating the traditional barriers to entry. When “services” are imprisoned inside proprietary applications, they can be prohibitively expensive to duplicate. But when – through SOA – standards-based services are made generally available, new players can tap them just as easily as established players. This means that the barriers to entry in an industry will no longer be technical and costly. They'll be only imagination, creativity, and nerve – qualities in no short supply among millions of hungry entrepreneurs.

Is this utopia, or just a level playing field? If you take the logic to its extreme, you can easily envision a global economy in which standards-based services will be available to everyone, and those who come out on top will be the ones who assemble and reassemble these services in the most creative ways in response to – or better yet, in anticipation of – emerging business opportunities. In this sense, SOA can be seen as an expeditor of the more perfect competitive equality to which markets are always tending. The Internet in its first incarnation spawned many new competitors – eBay, Amazon, and Google, to name a few – mainly because it let people connect to machines (and information) in entirely new

ways. This next incarnation – SOA – will let machines connect to other machines in entirely new ways – spawning yet another wave of innovative business designs and companies.

Those are some pretty big picture ideas, and I think they'll give you the sense that you had better SOA-enable your infrastructure to be able to participate in this new economic order. But where exactly do you start – and though SOA might suit many if not most companies, how can you be sure that it suits yours?

There's a great deal of detail on all this in a white paper, but here are guiding principles in brief. To get started:

1. Focus on a real business problem (preferably one with revenue potential)
2. Start small
3. Find skills
4. Think long-term (in other words, don't "go" big. Just "think" big)

This last point is exceedingly important, for, as we discuss in a companion white paper, SOA – a practical guide to measuring return on that investment, the real payoff from SOA comes when, after you endow your infrastructure with it, you keep creating powerful new applications at comparatively little cost. Think of it as modular, or atomic, business.

And what kinds of companies most benefit from SOA. Well, consider these attributes:

1. An extensive dynamic set of trading partners
2. Multiple software applications and interfaces
3. Products and services with a large IT component
4. Frequently changing business processes
5. IT capability as a defining characteristic


6. The presence in your network of a buyer or supplier with disproportionate power

At least in our experience as a business partner to thousands of firms around the world, this list, to one degree or another, defines most of them. One misconception afoot is that small companies probably don't need SOA – because, it's alleged, their infrastructure isn't nearly as complex as larger firms'. That might be true, but to tap into the galaxy of service components increasingly available, they'll need SOA themselves. It's their ticket to broader integration – an opportunity to act big just as the first wave of the Internet offered. I'll amend the above list to declare this, unabashedly: there's really no firm out there that doesn't need SOA. Again, like the Internet, SOA will become a sine qua non to participate in your network, your industry, and the overall economy.

Remember, SOA is an investment in flexibility. Any good business strategy involves flexibility, and SOA, because it liberates the full potential of your IT by making it modular and more flexible, is at the enabling core of that strategy. I guarantee you that if your enterprise, no matter the size, isn't thinking about how to do SOA today, your competition is – maybe not your largest competitor now, but your future industry leader. ■

About the Author

Michael Liebow is the vice-president of Web Services and SOA for IBM Global Services and he views Web Services as the critical enabling technology in IBM's On-Demand vision. Michael has a broad background that makes him well suited to bringing new concepts to market. He has held a variety of sales, marketing, and management positions in a diverse range of industries, including the high-tech sector, consumer packaged goods, media and entertainment, and advertising.



BUILD RICH INTERNET APPS!

Rich Internet Applications with Adobe Flex & Java

Written by Yakov Fain, Dr. Victor Rasputnis and Anatole Tartakovsky

"The authors have been key contributors to Flex's success via their participation in our beta programs, posts to community forums, public presentations, and blog postings...There's a lot to learn, but Yakov, Victor, and Anatole have done an excellent job introducing you to everything you need to know to build a robust application."

— Matt Chotin, Adobe, Product Manager

Order Online at TheRIABook.com and get

40% OFF

Regular Bookstore Price!

books.sys-con.com
NOW SHIPPING!
BOOKS

SYS-CON BOOKS books.sys-con.com
from the World's Leading i-Technology Publisher © COPYRIGHT 2007 SYS-CON MEDIA



Solve Performance Problems with FastSOA Patterns

The appropriate solution

Printed with permission from Morgan Kaufmann, a division of Elsevier. Copyright 2007.

"FastSOA: The way to use native XML technology to achieve Service Oriented Architecture governance, scalability, and performance" by Frank Cohen.

For more information about this book and other similar titles, please visit www.mkp.com.

WRITTEN BY FRANK COHEN

Here we're going to show a FastSOA mid-tier service and data caching architecture applied in three real-world scenarios. The scenarios show how to accelerate SOA performance and mitigate performance problems through mid-tier service caching, native XML persistence, and mid-tier data transformation, aggregation, and federation.

Three Use Cases and the FastSOA Pattern

FastSOA is an appropriate solution for SOA performance and scalability challenges. Each use case shows how pure XML technology used in the mid-tier mitigates and solves performance and scalability problems and delivers flexibility unavailable with object and relational technology. While there are many (sometimes contradictory) definitions of SOA, most software developers and architects recognize and support SOA as a pattern built around consumers, services, and brokers. Figure 1 shows this relationship. The basic SOA patterns make sense for developers building services, Web Services, and composite applications and data services. The pattern lets a consumer who makes a request to learn the location and interface message schema of a service. The consumer binds to the service by sending a request message. The service returns a response

message to the consumer's request. The service makes its location known by publishing the ontology of its functions and interface message schema to the broker. SOA is an abstract architecture — for instance, SOA doesn't define a specific protocol such as SOAP in the Web Services standard — but most SOA designs use XML as the message format between consumer, broker, and service.

To understand the SOA pattern in practice, we'll look at three scenarios and show how FastSOA solves scalability, performance, and flexibility challenges in each.

- Accelerating service interface performance and scalability
- Improving SOA performance to access services
- Flexibility needed for Semantic Web, service orchestration, and services dynamically calling other services

Scenario 1: Accelerating Service Interface Performance & Scalability

Suppose a business operates a parts ordering service. The service provides a Web browser user interface to enter a new order and

learn the status of an existing order. Behind the UI is a SOAP-based Web Service using ebXML message schemas to track the status of the order from a vendor's legacy inventory system. The service stores orders and customer information in a relational database. Figure 2 illustrates a typical use case.

The use case begins with a customer placing an order. The service validates the order against the current inventory to make sure the part being ordered is in the parts catalog. The service stores the order until the company consolidates all the orders

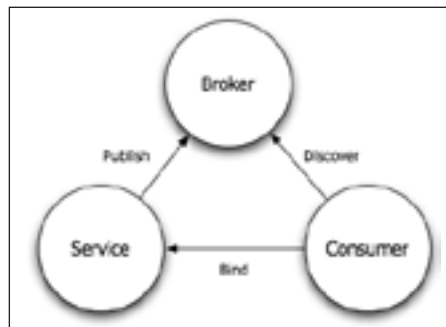


Figure 1: The basic SOA pattern.

in a nightly batch process with the parts vendor. The service ends the use case by checking the status of the order. Figure 3 illustrates an n-tier architecture often recommended in Java development to implement a parts ordering service.

The architecture divides into three parts: A presentation tier, an application tier, and a data tier. The presentation tier uses a Web browser with AJAX and RSS capabilities to create a rich user interface. The browser makes a combination of HTML and XML requests to the application tier. Also at the presentation tier is a SOAP-based Web Services interface so a customer system can access the parts ordering functions. At the application tier, an Enterprise Java Bean (EJB) or plain old Java object (POJO) implements the business logic to respond to the request. The EJB uses a Model, View, and Controller (MVC) framework — for instance, Struts or Tapestry — to respond to the request by generating a response Web page. The MVC framework uses an object/relational (O/R) mapping framework — for instance, Hibernate or Spring — to store and retrieve data in a relational database.

There are three problem areas that cause scalability and performance problems when using Java objects and relational databases in XML environments. Figure 4 illustrates these problems.

- XML/Java mapping requires increasingly more processor time as XML message size and complexity grow.
- Each request operates the entire service. Many times the user will check order status sooner than any status change is realistic. If the system kept track of the most recent response's time-to-live duration then it wouldn't have to operate all of the service to determine the most previously cached response.
- The vendor application requires the request message to be in XML form. The data the EJB previously processed from XML into Java objects now has to be transformed back into XML elements as part of the request message. Many Java-to-XML frameworks — for instance, JAXB, XMLBeans, and Xerces — require processor-intensive transformations. These frameworks challenge developers to write difficult and complex code to do the transformation.
- The service persists order information in a relational database using an object/relational mapping framework. The framework transforms Java objects into relational rowsets and does the joins among multiple tables. As object complexity and size grow, many developers have to debug the O/R mapping to improve speed and performance.

To give you an idea of the extent of the problem, consider the performance advantage of using native XML technology to respond to service requests. Figure 5 contrasts the performance difference. The results in Figure 5 contrast native XML technology and Java technology to implement a service that gets SOAP requests. The test varies the size of the request message among three levels: 68 kilobytes, 202 kilobytes, and 403 kilobytes. The test measures the roundtrip time it takes to respond to the request at the consumer. The test results are from a server with dual-Xeon 3GHz processors running on a gigabit-switched Ethernet network. The code was implemented in two ways:

1. **FastSOA technique.** Uses native XML technology to provide a SOAP service interface. Raining Data TigerLogic's XML query (XQuery) engine was used to expose a socket interface that gets



Figure 2: A typical use case for a parts ordering service

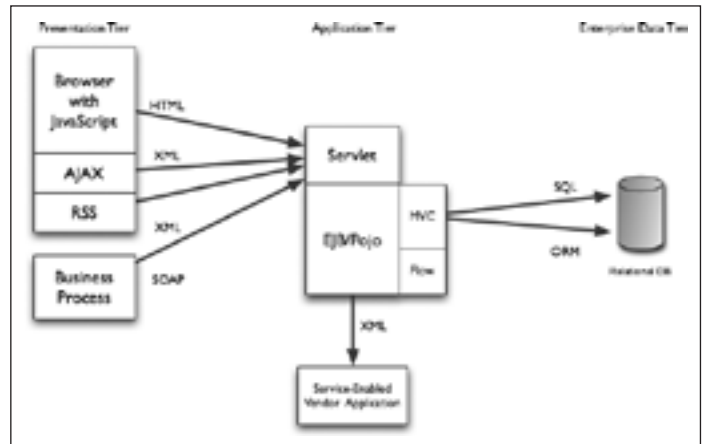


Figure 3: Building the parts ordering service using a domain pattern

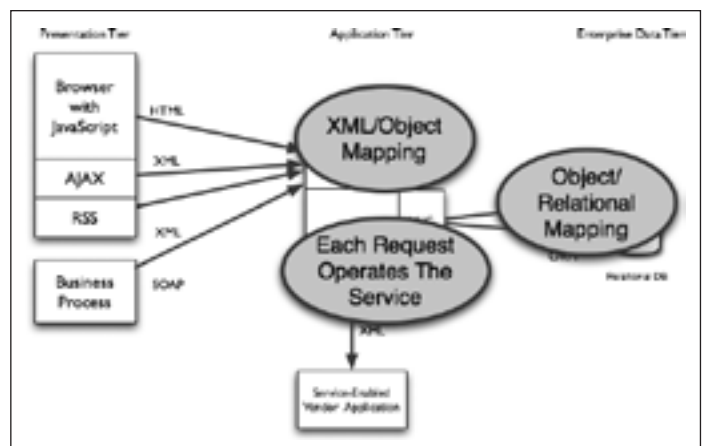


Figure 4: The source of scalability and performance problems – all that mapping and transformation

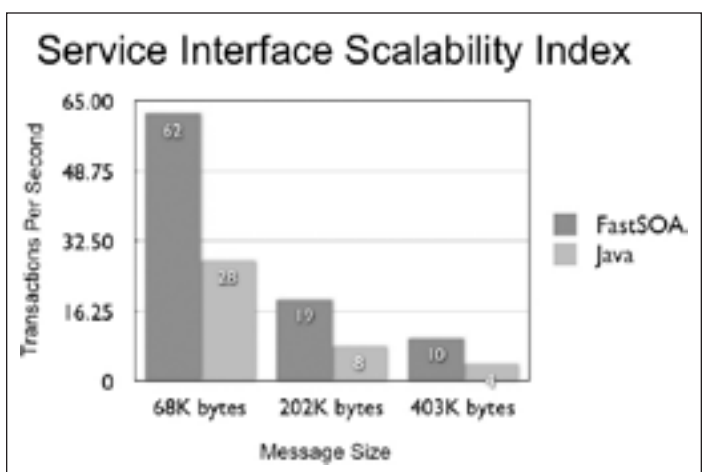


Figure 5: Contrasting service interface performance between techniques

the SOAP message, parses its content, and assembles a response SOAP message.

2. **Java technique.** Uses the SOAP binding proxy interface generator from a popular commercial Java application server. A simple Java object gets the SOAP request from the binding, parses its content using JAXB-created bindings, and assembles a response SOAP message using the binding. The results show a 2-2.5 times performance improvement when using the FastSOA technique to expose service interfaces.

The FastSOA method is faster because it avoids many of the mappings and transformations that are done in the Java binding approach to work with XML data. The greater the complexity and size of the XML data, the greater the performance improvement.

FastSOA is equally applicable to improving the performance of SOA application requests that require access to data. FastSOA implements a mid-tier cache to commonly accessed data. Caching is a powerful and proven technique for database systems. For instance, RDBMS tools vendors perform caching strategies at three points in an SOA environment, as illustrated in Figure 6.

In the presentation tier, a Web cache provides cached data to Java Server Faces (JSF) dynamic pages. At the application tier, caching at the object/relational mapping (ORM) and in the connection layer to the relational database (RDBMS) is recommended using an in-memory cache. Caching in this environment works well to mitigate relational performance problems.

These caching techniques should, but don't, take advantage of the unique features of XML data. Many SOA XML messages include

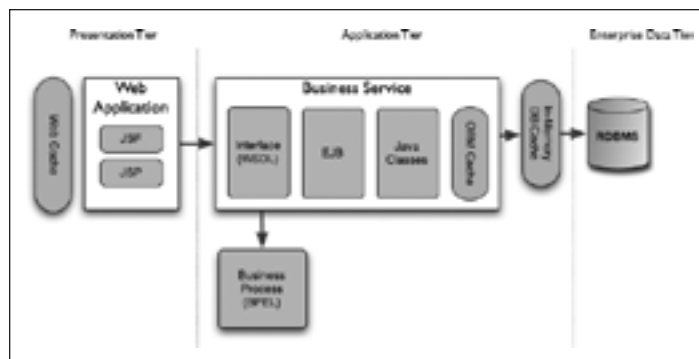


Figure 6: Mitigating relational performance through caching

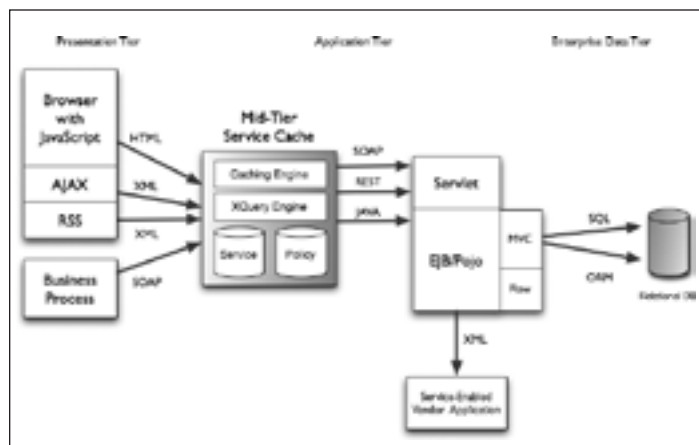


Figure 7: Using XML technology to provide service acceleration through caching

a time-to-live value in the message itself. This gives the cache intelligence about the lifetime of data that's usually unavailable in generic data caching approaches. To achieve this, cache intelligence requires a programming language and persistence engine of its own.

Figure 7 shows a FastSOA architecture using native XML technology to provide a service interface, accelerate service performance by caching response data, and implement flexible and rapidly changed policies to operate the cache engine.

The advantage to using the FastSOA architecture as a mid-tier service cache is its ability to store any general type of data, as well as its strength in quickly matching services with sets of complex parameters to determine efficiently when a service request can be serviced from the cache. The FastSOA mid-tier service cache architecture does this by maintaining two databases.

- The service database holds the cached message payloads. For instance, the service database holds a SOAP message in XML form, an HTML Web page, text from a short message, and binary from a JPEG or GIF image.
- The policy database holds units of business logic that look into the service database contents and make decisions on servicing requests with data from the service database or passing through the request to the application tier. For instance, a policy that gets a SOAP request validates security information in the SOAP header to validate that a user can get previously cached response data. In another instance, a policy checks the time-to-live value from a stock market price quote to see if it can respond to a request from the stock value stored in the service database.

FastSOA uses the XQuery data model to implement policies. The XQuery data model supports any general type of document and any general dynamic parameter used to fetch and construct the document. Used to implement policies, the XQuery engine lets FastSOA efficiently assess common criteria of the data in the service cache, and the flexibility of XQuery allows for user-driven fuzzy pattern matches to efficiently represent the cache.

FastSOA uses native XML database technology for the service and policy databases for performance and scalability reasons. Relational database technology delivers satisfactory performance to persist policy and service data in a mid-tier cache provided the XML message schemas being stored are consistent and the message sizes small. To understand this in greater depth, consider the following results from a comparison of native XML database technology to relational databases.

The test runs multiple test cases where each test case varies the number of concurrent requests made to the database host and varies the size of the XML message. The test environment makes requests to a relational database and a native XML database. For the relational database the test used an XML CLOB field type. The use case is modeled around a mid-tier cache's need to rapidly persist and query an unknown quantity of data with multiple and unknown schemas.

The test environment monitors the time it takes to get a response at the consumer. The test produces a report showing transactions per second (TPS) at each level of concurrent requests and message payload sizes. Figure 8 shows a scalability index report for querying a database of stored XML documents.

In the query performance test, the native XML database starts at 45TPS and goes up to 186TPS while the relational database stays at 15TPS or less. Figure 9 compares performance characteristics while

inserting XML documents into the database. The insert document performance test shows that native XML database and relational database performances are evenly matched at 20 and 17TPS at the lowest number of concurrent requests. At 32 concurrent requests, the native XML database performs 3.4 times (48TPS/14TPS) as many inserts than the relational database performs.

These results are not meant to knock relational database technology out of the running for XML persistence, because there are undoubtedly a large number of optimizations that could be employed to improve performance. Instead, these numbers are meant to prepare you for the performance and scalability challenges in SOA environments that use an unknown variety of XML message sizes and schemas.

There's no gatekeeper to the number and form of XML message schemas in an SOA environment. Mid-tier caching strategies need the best-performing and most flexible database available that can handle multiple and unknown message schemas and data formats. The benefits to a business running a FastSOA mid-tier service cache include:

- Fewer CPU licenses for commercial application servers
- Less network overhead
- Improved performance as compared with other mid-tier cache architectures
- Advanced SOAP in-line processing for a two-three times performance improvement over binding proxies created with Java application server utilities
- More efficient relational-to-XML transformation processing

The next scenario shows how FastSOA is used as a service data cache to improve data retrieval performance.

Scenario 2: Improving SOA Performance to Access Services

Imagine that a business operates a portal for employees to sign up for a retirement plan, medical insurance plan, and other programs. The portal application interfaces to an external database using a JDBC driver to retrieve company news. It also interfaces using REST protocols (XML over HTTP) for employee benefits data from the human resources service. The portal permits single sign-in to these services by interoperating with the corporate directory using LDAP protocols. Figure 10 illustrates a typical use case.

The use case begins with an employee signing in. The portal application validates the user credentials. Validated users sign up for alerts to enable the system to send an email notification when new healthcare plans become available. The employee uses the portal to browse healthcare plans and choose a plan. The service ends the use case by confirming the plan choice.

Figure 11 illustrates an architecture often recommended in the Java development community to implement the employee portal. The architecture divides into three portions: a presentation tier, an application tier, and a data tier. The presentation tier uses a Web browser with AJAX and RSS capabilities to create a rich user interface. The portal application also presents data to systems in other departments using SOAP and Java interfaces. At the application tier, an Enterprise Java Bean (EJB) implements the business logic

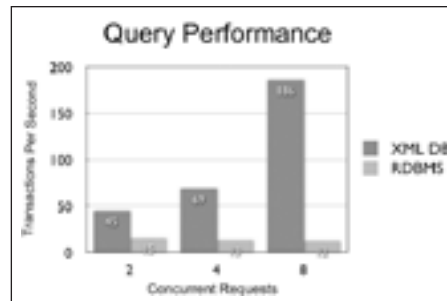


Figure 8: Comparing query performance between a native XML database and a relational database management system

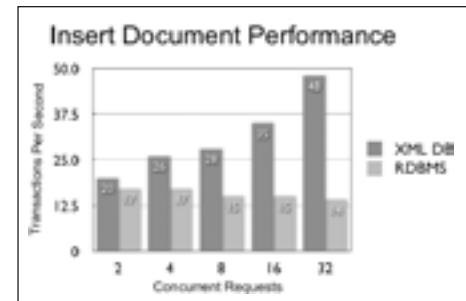


Figure 9: Comparing insert performance

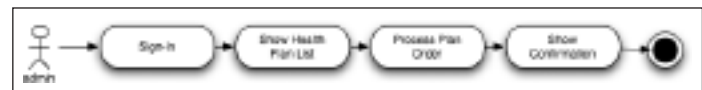


Figure 10: The employee portal use case

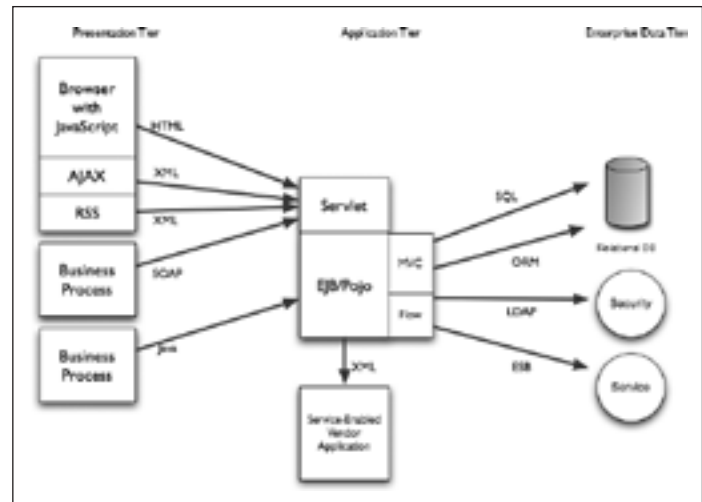


Figure 11: The employee portal built with Java and relational technologies

to respond to the request. The EJB interoperates with a relational database, security manager, and human resources service. Corporate mandates require the system to store the plans offered to the employees in its original XML form.

This architecture puts a lot of emphasis on the EJB. The EJB must produce aggregated views of the healthcare plans according to the user's position in the company. The views are assembled from XML data sources (the human resources system) and LDAP security provider.

There are three problem areas that cause scalability, performance, and developer productivity problems when using Java objects and relational databases in what is otherwise an XML environment in this scenario:

- **Slow object/relational mapping.** The views from the aggregated XML services need to be mapped into objects and then mapped into relational rowsets. This requires a lot of coding and processing.
- **Every request accesses the services and database slowly.** There are only so many types of employees. Yet the EJB assembles the views into the plan data each time a user requests a page.
- **Schema changes cause coding changes.** Every change to the human resources system message schema requires you to be right back in the code for the EJB.

Figure 12 shows an architecture that uses XML-centric technology to create aggregate views of data efficiently and rapidly from multiple data sources, accelerate data source performance by caching commonly used view data in the mid-tier, and implement flexible and rapidly changed policies to operate the data cache.

The mid-tier data cache stores XML, non-XML (such as binary), and other general types of data in two databases:

Direct views database. Holds views of the data from upstream data source providers. For instance, when the human resources healthcare plan services go off-line, the mid-tier data cache still services the last available view of the data. In another instance, the EJB may need a healthcare plan using a more modern schema and the direct views database holds a transformed healthcare plan from the human resources service emitting plans in an older schema. The direct views database may hold an aggregated view of two plans and an archive of how the plans have changed.

Aggregate views database. Holds data views that are intelligently composed as data is served to the EJB. For instance, in the employee portal scenario the aggregate views database stores the most recent plans viewed, keeps a list of popular plans, and keeps a list of high-quality plans as ranked by employee feedback. The mid-tier data cache uses a set of policies to determine the contents of the direct and aggregate views databases.

Each policy holds the business logic to look into upstream data sources and make decisions on which data to update and how frequently. The policy system is efficient at processing the business logic and storing the resulting XML views of the data from its use of native XML technology.

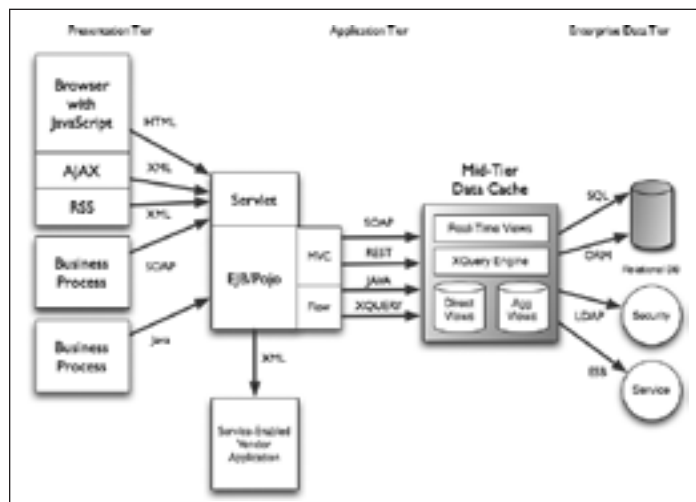


Figure 12: Implementing a data cache for aggregate views

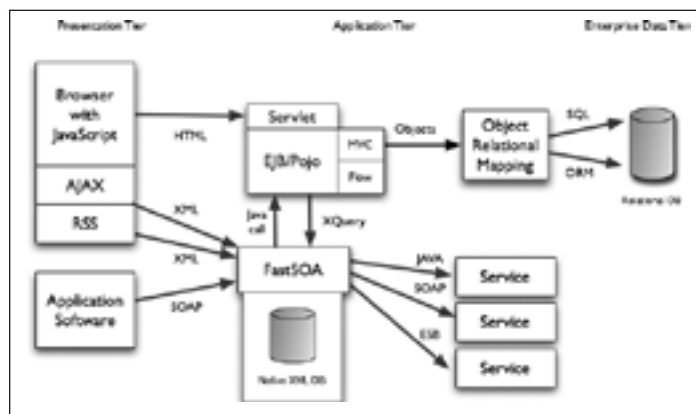


Figure 13: FastSOA in a Semantic Web environment

Choosing the FastSOA architecture for a mid-tier data cache in this scenario delivers a way to mitigate XML schema migration problems, reduce the amount of object-to-relational-to-XML mapping and transformation, and provides off-line data browsing capability. All of this comes without requiring you to be back in the EJB writing code.

The benefits to a business running a FastSOA mid-tier data cache include:

- Direct and aggregated views of the data model
- Real-time and near-time access between data and application tiers
- Two to 20 times performance advantage

In the next scenario, we'll show how FastSOA is used as a platform for building high-performance and scalable dynamic services.

Scenario 3: Flexibility Needed for Semantic Web, Service Orchestration, and Services Dynamically Calling Other Services

Over the past two years, the software development community has enjoyed a renaissance of creativity from new XML-based technology, including mashups (for instance, combining Google maps with photos from Flickr), AJAX for better user interfaces, and REST for easy application-to-application interoperability.

Much of this creativity pushes software development in the direction what Tim Berners-Lee espoused in the Semantic Web.

When services communicate with other services, XML is the interoperability standard. FastSOA has much to offer a software developer working in an XML environment. Figure 13 illustrates an architecture that adds fast, efficient, and flexible XML capabilities to an otherwise Java and relational database architecture.

The FastSOA architecture enables developers to write business logic using pure XML technology. The FastSOA architecture provides the XML interface to the existing Java and relational database systems. For instance, when a browser makes an XML request from an AJAX component, FastSOA gets the request, sees if the response is cached and still valid, and responds with the cached response.

If the request requires data provided by a Java object (in the EJB) then FastSOA makes a direct Java call to the object and method. FastSOA as a service interface and component development environment for Semantic Web applications brings the following business benefits:

- 100% native XML environment
- Avoiding object/relational/XML mapping reduces need for expensive application servers and network bandwidth
- Reduced software maintenance over time as message schemas

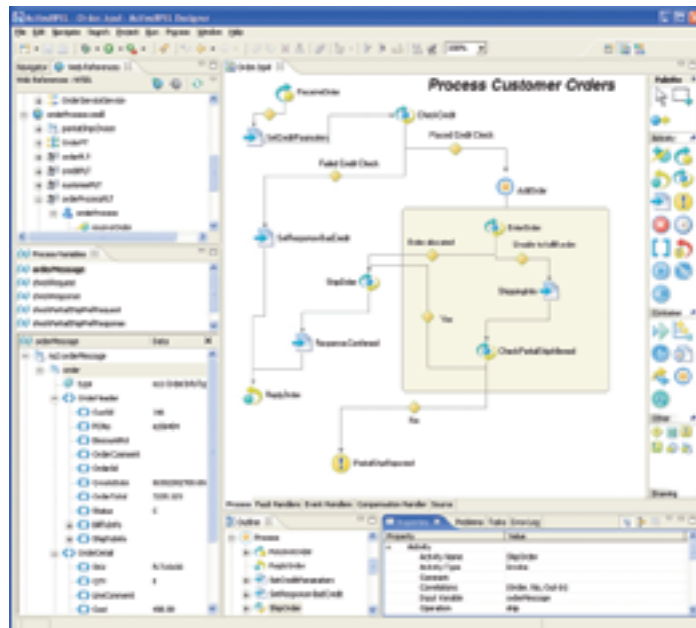
These three use cases show where FastSOA is an appropriate solution for XML performance and scalability challenges. We saw how native XML technology used at the mid-tier mitigates and solves performance and scalability problems and delivers flexibility unavailable with object and relational technology. The following are the business benefits for FastSOA:

- Solves SOA scalability problems for fast throughput and good scalability
- Works in existing infrastructure to avoid replacement costs
- Easy to customize with enterprise business processes using XQuery-based components
- Improves business agility and flexibility by maintaining interoperability and accelerating performance ■

About the Author

Frank Cohen is the principal maintainer of the popular TestMaker open source test utility and framework, and director of solutions engineering at Raining Data, publisher of the TigerLogic XQuery engine and native XML database.

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**

IF THIS IS YOUR STRATEGY TO GOVERN YOUR SOA - LET'S TALK

Software AG is a global leader in mission-critical software infrastructure solutions based on open standards.

With technology from Software AG, governance becomes a fundamental SOA principle:

- Improve transparency with a well-managed SOA
- Control your services with enforced policies
- Realize your SOA with confidence

More than 3,000 customers in 70 countries trust our solutions to maximize the value of their IT.

SOFTWARE AG
TAKE THE LEAD - LET'S TALK!

Contact us at www.softwareag.com