No. 1 *i*-Technology Magazine in the World

JDJ

JDJ.SYS-CON.COM VOL.12 ISSUE:1

Mediation and the Man in the Middle

Patterns for designing scalable and robust user-interfaces

RETAILERS PLEASE DISPLAY
UNTIL MARCH 31, 2007
$5.99US $6.99CAN
02>
0 09281 01751 6

PLUS...

- EJB 3 Transactions
- Configuring WebLogic Server 9.x JDBC
- What Is SCA?
- Performance Management 101 for WebLogic Portal

Altova® MapForce® 2007 – The premier data integration and Web services implementation tool.

ALTOVA® mapforce® 2007

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

ALTOVA®

JAVA DEVELOPER'S JOURNAL

Editorial Board
Java EE Editor: **Yakov Fain**
Desktop Java Editor: **Joe Winchester**
Eclipse Editor: **Bill Dudney**
Enterprise Editor: **Ajit Sagar**
Java ME Editor: **Michael Yuan**
Back Page Editor: **Jason Bell**
Contributing Editor: **Calvin Austin**
Contributing Editor: **Rick Hightower**
Contributing Editor: **Tilak Mitra**
Founding Editor: **Sean Rhody**

Production
Associate Art Director: **Tami Lima**
Executive Editor: **Nancy Valentine**
Research Editor: **Bahadir Karuv, PhD**

To submit a proposal for an article, go to
http://jdj.sys-con.com/main/proposal.htm

Subscriptions
For subscriptions and requests for bulk orders, please send your letters to Subscription Department:

888 303-5282
201 802-3012
subscribe@sys-con.com

Cover Price: $5.99/issue. Domestic: $69.99/yr. (12 Issues) Canada/Mexico: $99.99/yr. Overseas: $99.99/yr. (U.S. Banks or Money Orders) Back Issues: $10/ea. International $15/ea.

Editorial Offices
SYS-CON Media, 577 Chestnut Ridge Rd., Woodcliff Lake, NJ 07677
Telephone: 201 802-3000 Fax: 201 782-9638

Java Developer's Journal (ISSN#1087-6944) is published monthly (12 times a year) for $69.99 by SYS-CON Publications, Inc., 577 Chestnut Ridge Road, Woodcliff Lake, NJ 07677. Periodicals postage rates are paid at Woodcliff Lake, NJ 07677 and additional mailing offices. Postmaster: Send address changes to: Java Developer's Journal, SYS-CON Publications, Inc., 577 Chestnut Ridge Road, Woodcliff Lake, NJ 07677.

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

Newsstand Distribution Consultant
Brian J. Gregory/Gregory Associates/W.R.D.S.
732 607-9941, BJGAssociates@cs.com

Which Is More Important – Saddam Hussein or Google?

Jeremy Geelan

The year 2006 in which YouTube became culturally ubiquitous, Flash video became the de facto Internet video standard of the Web, Microsoft beta-launched Vista, and the Wii entered our lives – was also memorable for one or two other real-world events such as the hanging of Saddam Hussein, prompting the obvious question: Is the progress of *i*-Technology front-runners like Google and eBay more, or less, important than the trial and execution of Saddam.

The difficulty of a working life spent examining and indeed celebrating the vibrant and energetic world of Internet technologies (*i*-Technology) is that there is always a risk of allowing the real – as opposed to the virtual – world to slide into relative insignificance. E-mails risk seeming more important than reality, and the birth of mere Web sites takes on the gravity of much more substantive and world-changing events.

How do we gauge the relative importance of a war criminal's extinction and a Web site's birth? What, in short, matters most: the world of geopolitics, wars, and globalization, or the far more peaceful, equally globalized world of the Internet?

It is a problem set that can be addressed in different ways. In the case of Bill Gates, for example, he has committed greater sums of money made through i-Technology than any man alive (specifically, an endowment fund now worth US $24 billion) to bringing innovations in health and learning to the global community, a cause that has attracted in addition the substantial fortune of Warren Buffett – who gave the Bill & Melinda Gates Foundation 85% of his fortune.

But what of the rest of us, those of us with a current net worth less than $53 billion and a 2006 salary plus bonus of $966,667? Should we just rejoice that 2006 has seen the liveliest array of Web 2.0 start-ups since the bursting of the Silicon Bubble? Or should we take more interest in the prospects for global peace, or lack of it, in a real world that doesn't necessarily reflect the same sense of order and indeed hope as the World Wide Web?

> "The difficulty of a working life spent examining and indeed celebrating the vibrant and energetic world of Internet technologies (i-Technology) is that there is always a risk of allowing the real – as opposed to the virtual – world to slide into relative insignificance."

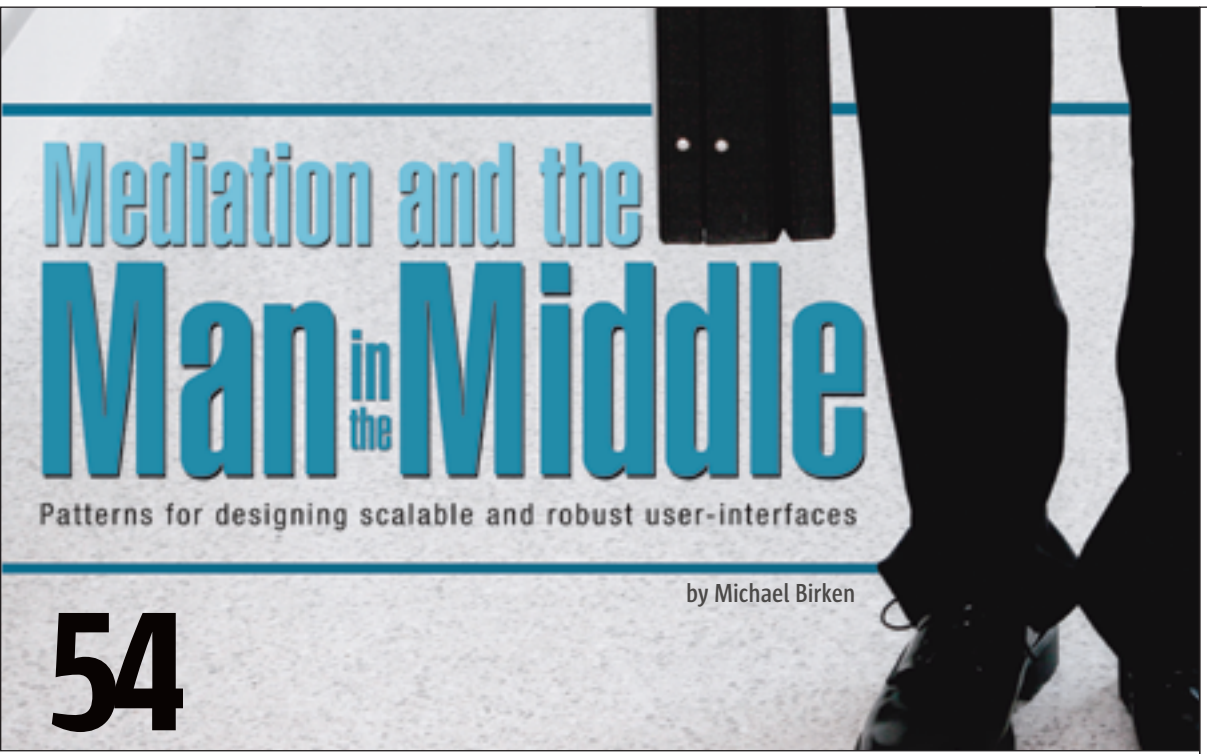
Jeremy Geelan is Sr. Vice-President, Editorial & Events of SYS-CON Media. He is Conference Chair of the AJAXWorld Conference & Expo series and of the "Real-World Flex" One-Day Seminar series. From 2000-6, as first editorial director and then group publisher of SYS-CON Media, he was responsible for the development of all new titles and i-Technology portals for the firm, and regularly represents SYS-CON at conferences and trade shows, speaking to technology audiences both in North America and overseas. He is executive producer and presenter of "Power Panels with Jeremy Geelan" on SYS-CON.TV, and is actively helping build out the AJAXWorld brand as well as developing entirely new Conferences and One-Day Seminars for SYS-CON Media & Events.

jeremy@sys-con.com

JANUARY 2007 VOLUME:12 ISSUE:1

JDJ contents

JDJ Cover Story



Features

EJB 3 Transactions
by Raghu R. Kodali and Jonathan Wetherbee

What Is SCA?
by Simon Laws, Haleh Mahbod, and Raymond Feng

FROM THE EDITOR
Which Is More Important – Saddam Hussein or Google?
by Jeremy Geelan 3

VIEWPOINT
Change Is Good!
by Nigel Cheshire 6

INDUSTRY COMMENTARY
Building the Right Project Team
The rule of five
by Robert Shinbrot 12

ARCHITECTURE
Performance Management 101 for WebLogic Portal
An introduction
by Rini Gahir 14

CODE
The Paradox of Writing Perfect Code
Static code analysis versus Santa Claus and the Easter Bunny
by Ben Chelf 20

WEBLOGIC
Configuring WebLogic Server 9.x JDBC
Data source connections
by Deepak Vohra 26

JAVA
Are Vendors Becoming More in Charge of Java Enterprise Edition...
...or is Sun losing control over Java EE?
by Andrei Iltchenko 34

DESKTOP JAVA VIEWPOINT
Software Should Be More Hard Wearing
by Joe Winchester 46

WEB SERVICES
Enterprise Mashup Services
Part 1: Real-World SOA or Web 2.0 Novelties?
by Ric Smith 48

JSR WATCH
Ringing in the New Year
by Onno Kluyt 60

JDJ (ISSN#1087-6944) is published monthly (12 times a year) for $69.99 by SYS-CON Publications, Inc., 577 Chestnut Ridge Road, Woodcliff Lake, NJ 07677. Periodicals postage rates are paid at Woodcliff Lake, NJ 07677 and additional mailing offices. Postmaster: Send address changes to: JDJ, SYS-CON Publications, Inc., 577 Chestnut Ridge Road, Woodcliff Lake, NJ 07677.

Nigel Cheshire
Guest Editor

Change Is Good!

In an article in the October edition of the FTP Webzine "Upside" Peter Varhol laments the trend toward per-developer metrics in the software development process. "Individual developer data is stored and available to be manipulated in less than honorable ways," he says, "and there are people in enterprises who know how to take advantage of such information for their own purposes."

Yes Peter, those people are called "managers" and their purposes are called "management." It's what they are supposed to do.

It's high time this industry grew up and, I'm afraid, growing up means opening up and letting managers see what developers are up to on a day-to-day basis. Earlier in his piece Varhol talks about the "bubble world" that developers have created for themselves by encouraging the view that software development is mysterious, highly complex, and technical.

Peter Varhol isn't the only person who seems change-averse in this respect. Other industry commentators have weighed in on the issue too. Linda Westfall is president of The Westfall Team, a consultancy that specializes in software metrics and software quality engineering training. "Don't measure individuals" she says in her 2005 paper on software metrics. "The state-of-the-art in software metrics is just not up to this yet." To be fair, she goes on to explain that what she's really talking about is individual productivity metrics measured in "lines of code per hour." True, that wouldn't seem to be a useful metric. But that doesn't invalidate the idea of looking at *any* metrics on a per-developer basis.

Sam Guckenheimer of Microsoft, in his book *Software Engineering with Microsoft Visual Studio Team System*, says: "Using metrics to evaluate individual performance is horribly counterproductive."

Joel Spolsky says, "FogBUGZ does not provide individual performance metrics for people."

So what's the deal? Why is everyone so against the idea of measuring the performance of individual developers? After all, in just about any other industry, you'll find per-individual metrics being applied without so much as a second thought.

Take sales for example. Culturally, you're unlikely to find any two groups in a company more diverse than sales and software development. But actually, these two departments have more in common than you might think. Both work in teams, but are highly dependent on the performance and contribution of individual members. In fact, oftentimes, the contributions of one or two "superstars"

–continued on page 10

> "It's high time this industry grew up and, I'm afraid, growing up means opening up and letting managers see what developers are up to on a day-to-day basis"

Nigel Cheshire is CEO of Enerjy Software, a division of Teamstudio Inc. He oversees product strategy and has been driving the company's growth since he founded it in 1996. Prior to founding Teamstudio, Inc., Nigel was co-founder and principal of Ives & Company, a CRM solutions consultancy. He holds a Bachelor of Science degree in computer science from the University of Teesside, England.

nigel_cheshire@enerjy.com

SYS-CON MEDIA

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

President and COO:
Carmen Gonzalez *carmen@sys-con.com*

Senior Vice President, Editorial and Events:
Jeremy Geelan *jeremy@sys-con.com*

Advertising

Vice President, Sales and Marketing:
Miles Silverman *miles@sys-con.com*

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

Advertising Sales Manager:
Andrew Peralta *andrew@sys-con.com*

Associate Sales Manager:
Kerry Mealia *kerry@sys-con.com*
Lauren Orsi *lauren@sys-con.com*

Editorial

Executive Editor:
Nancy Valentine *nancy@sys-con.com*

Production

Lead Designer:
Tami Lima *tami@sys-con.com*

Art Director:
Alex Botero *alex@sys-con.com*

Associate Art Directors:
Abraham Addo *abraham@sys-con.com*
Louis F. Cuffari *louis@sys-con.com*

Web Services

Information Systems Consultant:
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*

Customer Relations

Circulation Service Coordinator:
Edna Earle Russell *edna@sys-con.com*

Speed. Simplicity. Style.

download the FREE trial now!

WINDOWS® FORMS ASP.NET WPF JSF

grids scheduling charting toolbars navigation menus listbars trees tabs explorer bars editors

> The important thing is that we should think carefully about what behavior we want to encourage and then put mechanisms in place to capture the metrics to track our progress"

–*continued from page 6*

overshadow the contributions of everyone else put together. Both work for extended periods of time to deliver an outcome. And both are notoriously difficult to manage!

But there's one significant difference between these two groups. Sales teams, especially inside sales teams, have clear targets, not just for the outcome (dollars and cents) but also for their *behavior* on a day-to-day basis. I'll always remember one of the most successful sales managers I've ever known early in my career telling me the secret to her success, "Manage behavior; reward results." In the case of the inside sales team, the way we manage behavior is by looking at call metrics: the number of dials, total talk time, average call duration, etc. These are measures of the desired behavior of the team.

So, what's the desired behavior of the development team and what metrics can we put in place to measure it?

Well, Linda Westfall is right about one thing: we don't simply want to measure lines of code per hour. That's not a measure of productivity; it simply encourages verbosity! But perhaps we do want to measure compliance to coding standards. We might also want to measure unit test coverage and density, code complexity, reuse effectiveness, and defect rates.

The important thing is that we should think carefully about what behavior we want to encourage and then put mechanisms in place to capture the metrics to track our progress. But here's an important and rather subtle distinction between different types of metrics. Often, the thing that makes metrics, especially per-developer metrics, seem bad to so many people is that they when they think about metrics, they're thinking about *prescriptive* metrics. Prescriptive metrics carry with them a target level of achievement that defines success. The problem with prescriptive metrics is that they encourage slavish devotion to achievement of just that metric, sometimes at the cost of rational thought.

Joel Spolsky again: "As soon as you start measuring people and compensating people based on things like this, they are going to start optimizing their behavior to optimize these numbers, and not in the way you intended."

Coming back to our sales team example, if you just set a target for total talk time, without measuring the number of key contacts that the salesperson talks to, you encourage the wrong type of behavior – talking at length with someone who's never going to buy anything, for example.

The same thing applies in the software development world. It's generally accepted that 100% code coverage for unit tests isn't a good objective. Not only is it potentially time-consuming to squeeze the last few percentage points of coverage out of an application, but this is also the sort of thing that could really encourage the wrong behavior. Exception handlers are usually the hardest parts of the application to unit test. So what's the easiest way to increase your coverage percentage? Get rid of your exception handlers! That way you don't have to figure out how to test them.

Contrast that approach with the concept of *descriptive* metrics. As the name suggests, these are metrics that we just report, without any set notion of what's a "good" number in absolute terms.

To pick up the sales team example again the number of calls per day and total call time are descriptive metrics. We don't reward the team based on these metrics; we've already seen how that can lead to counterproductive behavior. However, if we observe a significant drop in sales by an individual then these metrics can provide a clue to the cause of the problem. Similarly, if a developer has consistently higher bug rates than his or her colleagues and lower unit test coverage then it's clear what action needs to be taken to bring that developer up to standard.

Here's the thing. This software development industry of ours is still pretty young. We're still figuring some of this stuff out. Blindly trying to apply manufacturing-style metrics to the software development process won't work. They are two different processes, and should be treated as such. But that doesn't mean we shouldn't be striving to apply some metrics and working hard to figure out what the metrics are and how to manage them.

So, to Mr. Varhol's target audience I say fear not. Embrace change. Don't stick your head in the sand and say metrics are a bad thing. Let's work together to figure out what we can measure, and how we can use those things to manage ourselves better. After all, our customers deserve better.

top **MISCONCEPTIONS** that drive

Meet the most misunderstood developer team in the world.

our Crystal Reports dev team crazy

Crystal Reports® is too expensive. Actually, the developer edition is just $595[1] USD (or upgrade for only $315[1]). Complimentary Crystal Assist support[2] provided with purchase.

Crystal Reports doesn't include a free runtime license. Not true, the developer edition includes a free runtime license[3] for each component engine.

Getting reports on the web is complex. False, the developer edition includes crystalreports.com[4] and Crystal Reports Server[5] to speed and simplify web reporting deployments.

Crystal Reports only works in Windows®. Not quite, whether you need to create or deploy reports on Windows, Linux or Unix, we have a Crystal Reports technology for you.

Find out more at: www.businessobjects.com/devxi/misunderstood

Business Objects™

1 Suggested retail price. 2 Complimentary access to support engineers and self-help. 3 Includes an unlimited runtime license for internal use of .NET, Java, and COM engines. 4 Includes ten named user licenses. 5 Includes five named user licenses.

Building the Right Project Team

by Robert Shinbrot

The rule of five

When building the right project team to complete a custom solution there are many forces at work. These include business drivers, technical drivers, and organizational and political motivations. Regardless of the business or organization there are three basic rules to follow in building a team to deliver a technical solution. The first is to involve the business before the team is even assembled. Each organization has certain technology standards that govern specific tools and products that can be used on a given project. These standards need to be considered and coordinated within "governance" management when architecting the solution. The third is the driving element that will let you successfully implement any medium or large-scale project and that's to follow "The Rule of Five." The Rule of Five is the basis for choosing the right number of people to be on your project team, and if you follow this rule your team will deliver the project on time and on budget.

Assume you are the project manager for a newly selected

technical implementation for a specific line of business that you're familiar with. You've been chosen for this effort because of the confidence that both the business and technology departments have in you to get the job done. Your past record speaks for itself and you have an opportunity to select people currently in your company as well as augment the team with new hires and consultants. Whether your project requires a medium or large-scale project team you should plan your team accordingly. Always plan in five-person team increments.

1. ***Business/Technical Lead*** – This person should be someone who understands the business requirements very well and hopefully was one of the main authors of the "Business Requirements Document." This individual should be technically very strong, but doesn't necessarily know all the technologies that have to be deployed to implement the system.
2. ***Technical Architect*** – This person is responsible for designing the technical framework in which the entire system will be built. He should be intimately familiar with all the technologies required to deliver the system and be mindful of all governance requirements in your organization. This is the lead person who will insure the technical success of the project.
3. ***Data Analyst*** – This person should be knowledgeable about all of the data ele-

> "A team that's too small or too big will deliver a project on time and over budget or late and over budget"

Robert Shinbrot has managed very large and complex projects over the last 20+ years within the Financial Services Community. Robert has led the Financial Services Consulting Practice first at Oracle and most recently at BusinessEdge Solutions for the last 5 years.

rshinbrot@businessedge.com

> When building your next project team think in terms of five and you'll be able to maximize your business and technical capability to deliver a solution on time and on budget"

ments required for system implementation as well as where the data currently resides in the organization and how to gain access to it. This person will coordinate all DBA requirements and work with the governance group in the organization as well as lead all logical and physical database design efforts.

4. ***Technical Programmer*** – This person will be responsible for coding parts of the system based on the direction of the **Project Manager**. For example, this person may be the front-end technical programmer.
5. ***Technical Programmer*** – This person will be responsible for coding parts of the system based on the direction of the **Project Manager**. For example, this person may be the application server programmer.

The Rule of Five focuses on any medium or large-scale project that will have a team composition based on these outlines. For example if the project is scoped to be a medium-sized project the team composition is one project manager and a team of five. If the project is scoped to be a large-scale project the team composition is one project manager and *n*(team of five) or either a 10-, 15-, or a 20-person team. It's rare that a project team is larger than 20, but the same rule holds. Keeping the team composition as listed above allows each team of five to work successfully on their portion of the project.

If your project team is greater than 20 people, the Rule of Five means a team of five to oversee all project activities and provide centralized project coordination or project governance as shown in Figure 1.

When determining who will be on your Rule of Five team follow the basic guidelines. The ***Business Technical Lead*** must be someone that is very senior, has direct contact with the business, and can resolve any outstanding business issues that come up. You should handpick this person from a small list of applicants. The ***Technical Architect*** must be very senior and preferably someone you've worked with before. He should have demonstrated superior knowledge in all technical aspects of the project and be hands-on at all times. The ***Data Analyst*** should be knowledgeable about ER tools and the SQL language being used in the project. This person should have worked on other projects in this group before to reduce the learning curve. This role is generally overlooked until late in the project. The ***Technical Programmers*** tend to be junior compared to other members of the team, but are focused on coding the application.

When building your next project team think in terms of five and you'll be able to maximize your business and technical capability to deliver a solution on time and on budget. A team that's too small or too big will either deliver the project on time but over budget or late and over budget. See if the "Rule of Five" works for you.

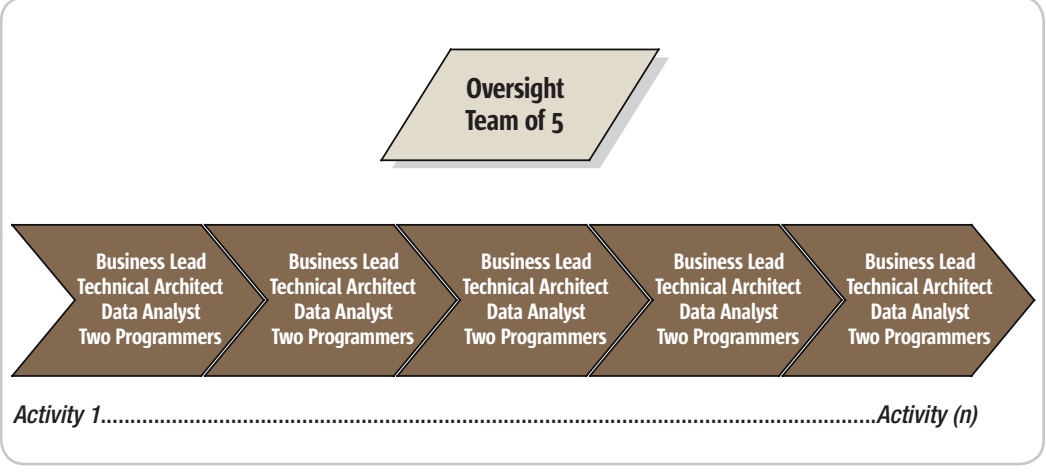


Figure 1 The rule of five

Performance Management 101 for WebLogic Portal

An introduction

by Rini Gahir

Even experienced Java Web developers are often surprised by how big a leap it is to develop a portal. The simple, slick interface that end users see belies the deep power and complexity provided by commercial products like BEA WebLogic Portal. This makes it extremely challenging to diagnose performance issues when portal applications go into production.

This article discusses the performance management challenges of WebLogic Portal and provides a starting point for tuning performance bottlenecks in portal applications. It assumes some familiarity with the terminology and functionality of WebLogic Portal.

A corporate portal allows a company to capitalize more effectively on its digital and human assets while presenting a first-class Web experience to its employees, partners and customers. For this reason, portal applications are becoming business-critical and must deliver reliable performance and scalability. BEA WebLogic Portal is a leading Java EE-based portal server that provides a robust solution for deploying and running portal applications.

Rini Gahir is a senior product marketing manager at Quest Software and has been working at the forefront of Java and Internet technologies in a wide variety of technical and business roles for over a decade.

rini.gahir@quest.com

WebLogic Portal Architecture

BEA WebLogic Portal combines a unified runtime framework, business services and lifecycle management technologies into a complete Web portal development and delivery platform. It is designed to scale to thousands of end users and support continuous changes.

Figure 1 shows the architecture of the WebLogic Portal hierarchy. When a portal is instantiated, it generates a taxonomy or hierarchy of portal resources known as the WebLogic Portal control tree. The control tree includes desktops, books, pages and portlets. As we'll see, the control tree provides one of the most important keys to understanding performance issues in portal applications.

The basic building block of the portal is the portlet, which is a small portal application, usually depicted as a small box in the Web page. They are reusable components that provide access to applications, Web-based content and other resources, and can access and display Web pages, Web services, applications and syndicated content feeds.

A portlet is developed, deployed, managed and displayed independent of other portlets. Administrators and end users can create personalized portal pages by choosing and arranging portlets, resulting in Web pages with content tailored for individuals, teams, divisions and organizations. Portlets rely on the portal infrastructure to access user profile information, participate in window and action events, communicate with other portlets, access remote content, look up credentials and store persistent data.

Since portlets are servlets, they share similar re-entrance and performance considerations. A single portlet instance (that is, a single instance of the portlet's Java class) is shared among all requesters. As there are a limited number of threads that process portlets and servlets, it is important for each portlet to do its job as quickly as possible so that response time for the whole page is optimized.

Understanding the Control Tree

The WebLogic Portal control tree represents all of the structural elements in the portal and acts as the infrastructure on which a new portal page will be built. A new control tree is created (or drawn from cache if it exists already) during control tree processing when the portal is instantiated. One of the most significant impediments to portal performance lies with the number of controls on a portal. The more portal controls (pages, portlets, buttons, etc.) you have, the larger your control tree

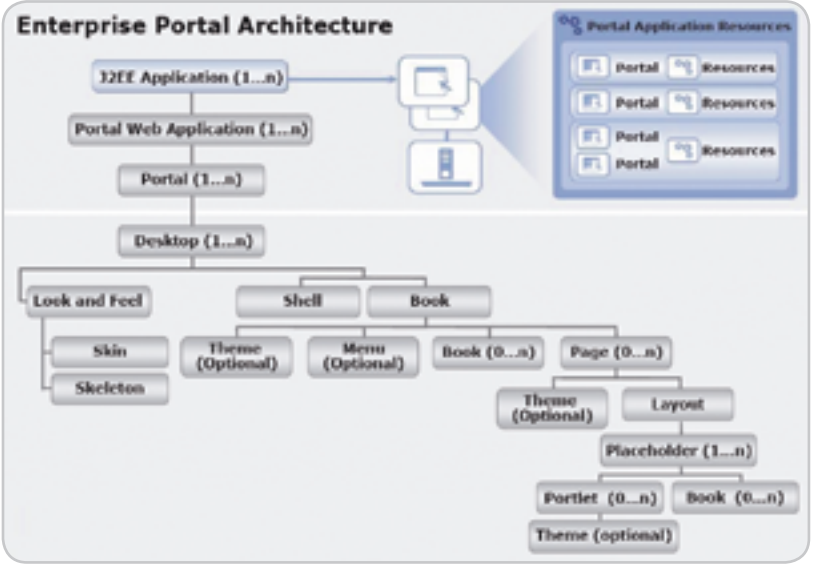


Figure 1 Hierarchy architecture of WebLogic Portal

MAKE ANSWERS TO PERFORMANCE PROBLEMS COME TO YOU.

OPNET Panorama offers powerful analytics for rapid troubleshooting of complex Java EE applications. Panorama quickly identifies how application, web, and database servers are impacting end-to-end performance. With Panorama, you can pinpoint the source of a problem, so time and money aren't spent in the wrong places.

The world's most successful organizations rely on OPNET's advanced analytics for networks, servers, and applications.

OPNET Panorama
Real-Time Application Analytics

www.opnet.com/panorama

OPNET®
Making Networks and Applications Perform™

OPNET Technologies, Inc. 7255 Woodmont Avenue, Bethesda, Maryland 20814 phone: (240) 497-3000 • e-mail: info@opnet.com • NASDAQ: OPNT

– and the longer it will take to render all the components.

Figure 2 shows a control tree generated for a typical portal. From the desktop and shell is created a main book and six sub-books, which in turn contain two pages each. Each page contains two portlets. So, in total, this portal has a minimum of 42 controls.

Once the control tree is built and all the instance variables are set on the controls, the tree must run through the lifecycle for each control before the portal can be fully rendered. The lifecycle methods are called in order. That is, all the init() methods for each control are called, followed by the loadState() method for each control, and so on in the order determined by the position of each control in the portal's taxonomy.

Running each control through its lifecycle requires some overhead processing time, which, when you consider that a portal might have thousands of controls, can grow exponentially. Thus, you can see that the larger the portal's control tree the greater the performance hit.

Monitoring Performance in WebLogic Portal

Portal performance is generally measured by the amount of time required to actually render that portal and all of its constituent parts when a visitor clicks an object that sends a request to the portal servlet.

The first challenge is simply in monitoring and measuring overall performance of the portal. Built-in management functionality really does not do a thorough enough job of this for the entire system, specifically the individual portal components (including portlets and other code run by the WebLogic Portal container), connections to any and all databases, transaction servers, mainframe systems and other back-end systems.

Whatever tool or tools you use needs to be able to:

- Monitor the complex, dynamic interactions taking place across the entire workflow and within individual processes,
- Present the resulting data in a clear, simple display that highlights problems (and where they occur in the portal workflow), and allow the administrator to quickly drill down – to individual portlets and transactions if need be – to the source of the problem, and
- Summarize overall performance as well as performance in the key portal workflow areas: portal servlet, control tree processing, JSP backing files, Java page flows, portlets, connections to back-end systems and portal services.

What to Monitor and Common Issues

There are several potential areas that can affect portal performance and availability. The following serves as a useful blueprint for what to monitor and the common problems that can manifest themselves.

Portal Request Response Time

Since portals are personalized Web applications, it is important to measure portal performance as an end user would experience it. By monitoring live transaction response times, the portal administrator can take proactive measures to address a problem before it impacts users and the business.

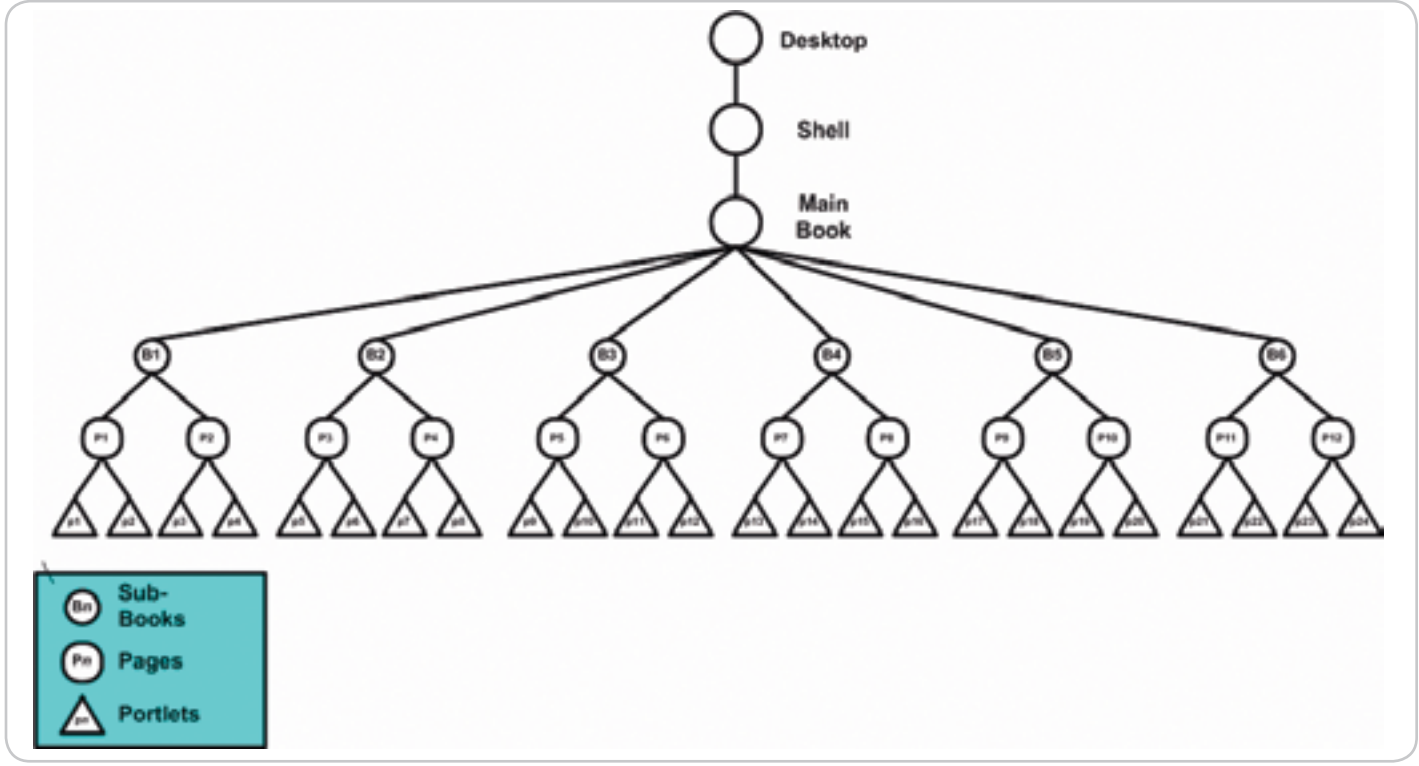
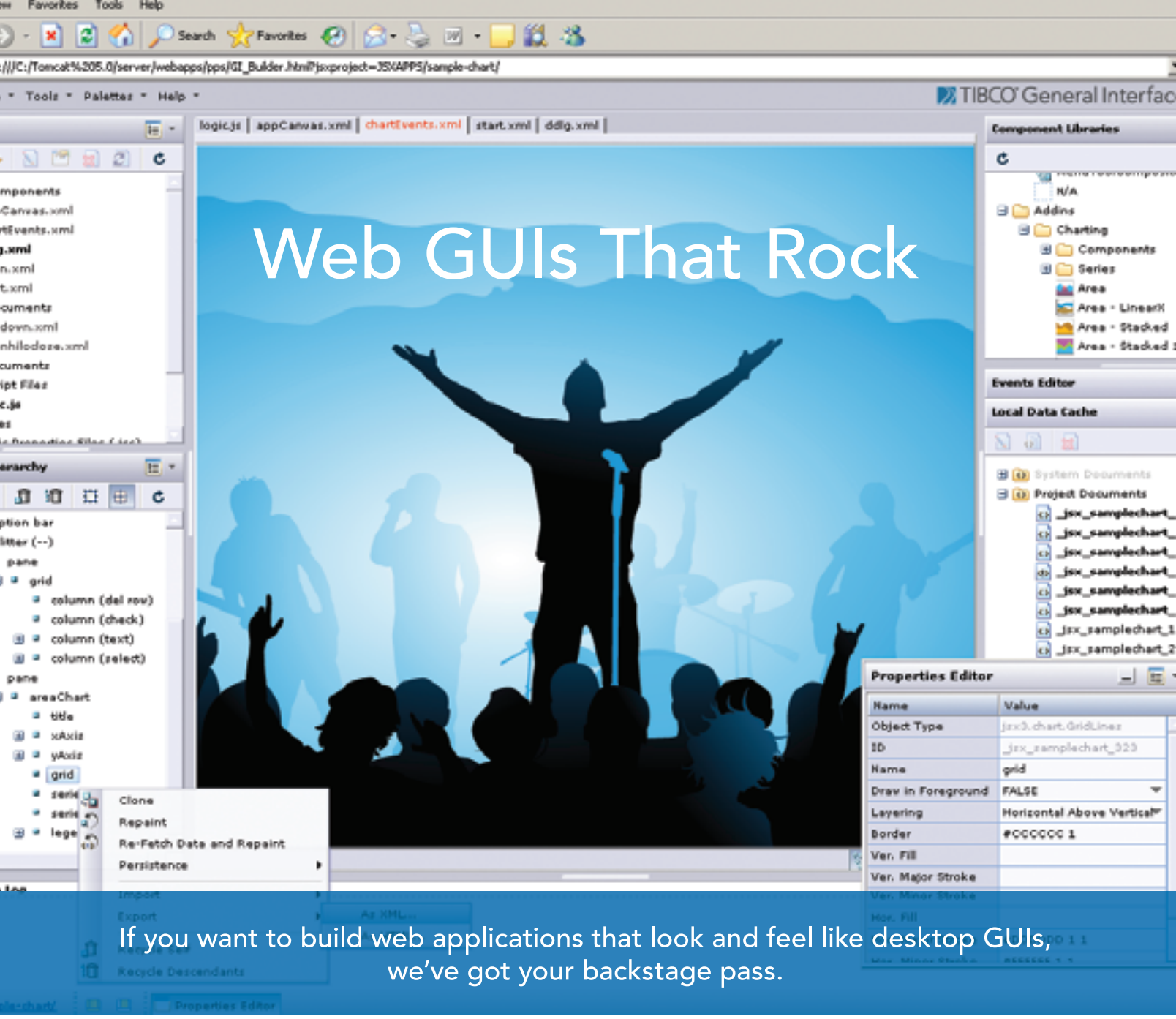


Figure 2 Typical control tree for a portal instance



Why are professional developers choosing TIBCO General Interface?
With more components and tools, TIBCO General Interface™ enables you to create AJAX applications that look and feel like desktop GUIs with astounding speed. No wonder it's the #1 independently rated AJAX Rich Internet Applications toolkit.

SOA and AJAX Rich Internet Applications are changing the way software is developed and deployed. TIBCO helps you make the shift from 3-tier to SOA-based computing to deliver astoundingly rich and agile solutions fast.

TIBCO®
The Power of Now®

Are *you* ready to rock? Learn more at http://developer.tibco.com/.

> A corporate portal allows a company to capitalize more effectively on its digital and human assets while presenting a first-class Web experience to its employees, partners and customers"

Control Tree Processing

Remember that the control tree represents all of the structural elements in the portal and acts as the infrastructure on which a new portal page will be built. Almost all of the elements in the user-interface design correspond to controls in this tree. Get visibility into the complex processes taking place within the control tree as well as its interactions with the "View" and "Control" elements of the portal. Figure 3 shows how a performance tuning tool might highlight a control tree performance issue.

Portlets

Applications, JSP-based portlets, Web Services or other available J2EE resources can all be exposed as portlets. If a performance slowdown occurs, application support personnel need the ability to determine quickly which individual portlets may be responsible. Within the portlet lifecycle, handling post back data and pre-rendering are processes that are especially important to monitor for performance.

Portal Framework Services

JSP backing files work in conjunction with JSPs, allowing separation of the presentation logic from the business logic. Always run before the JSPs, the backing files typically contain a great deal of custom rendering code (and additionally, some developers make callouts to back-end systems to fetch additional data to render). Poor performance is usually an indication of misbehaving custom rendering code.

In Java page flows, the page flow itself is entirely defined by the developer; slowness can usually be diagnosed by the author, and isn't usually caused by trouble with any back-end system. It may also be helpful to correlate the J2EE standard page flows to the portal control tree processing architecture to determine which page flow is tied to which desktop.

WebLogic Portal Services

The Entitlement system provides role-based authorization to individual portal resources. Entitlements are used heavily by all aspects of the portal, so any slowness impacts the whole system. Often, delayed responses and stalled threads are caused by trouble in the back-end systems supporting Entitlements, such as LDAP. Additionally, entitling too many objects at a fine level of granularity can overload the Entitlements system.

The Personalization service, implemented through advislets, modifies the information displayed in the portal preferences. Advislets can use many mechanisms — an internal rules engine, explicit personalization or even events. Overuse of the Personalization system overall is a common cause of performance problems.

The User Profile repository contains additional user information such as contact information. Often, delayed responses and stalled threads are caused by trouble in the back-end systems, such as a database used for supporting user profiles.

The Content Management API interfaces with a number of commercially available content management systems, such as Documentum. When stalled threads occur here, one of the first things to check is whether the back-end content system is performing normally.

Conclusion

We hope this has been a useful introduction to the performance challenges posed by WebLogic Portal applications. As enterprise portal offerings grow in popularity and complexity, so does the challenge of managing their performance and availability. With the proper tools and process, portal-based applications can be depended upon to consistently deliver their promised business value.

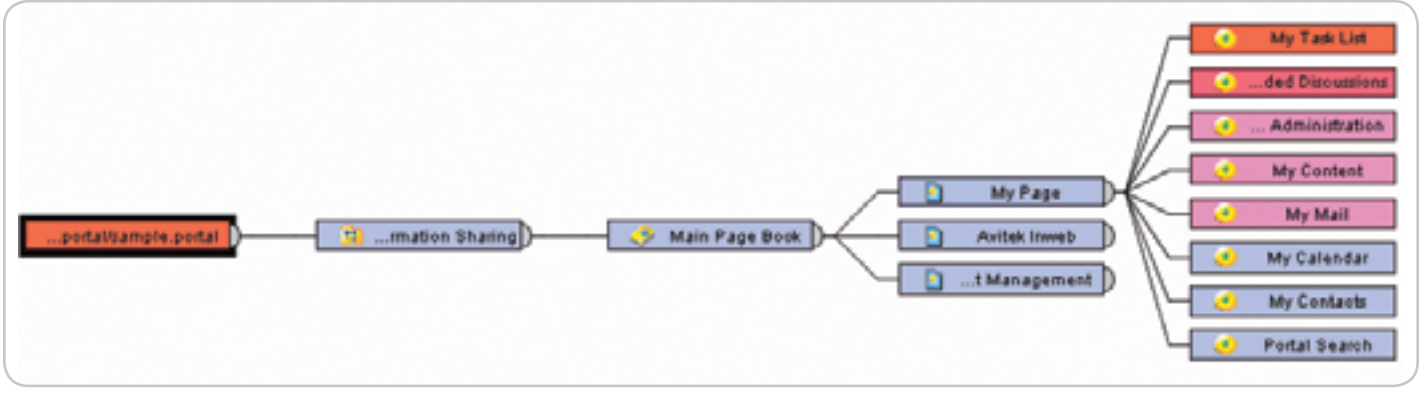


Figure 3 Visualizing control tree performance bottlenecks

BACKBASE

AJAX for Java

Backbase offers a comprehensive AJAX Development Framework for building Rich Internet Applications that have the same richness and productivity as desktop applications.

The Backbase AJAX Java Edition:

- is based on JavaServer Faces (JSF)
- runs in all major Application Servers
- supports development, debugging and deployment in Eclipse
- embraces web standards (HTML, CSS, XML, XSLT)

Download a 30-day Trial at www.backbase.com/jsf

i: www.backbase.com **t:** (866) 800-8996 **e:** sales-us@backbase.com

The Paradox of Writing Perfect Code

Static code analysis versus Santa Claus and the Easter Bunny

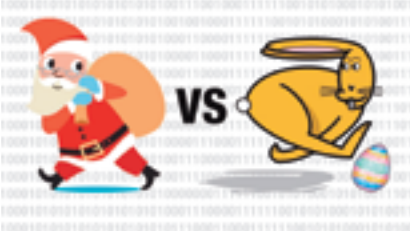
by Ben Chelf

Don't you love looking at a good piece of code? I'm talking about the kind of code where the design is so sound that the code practically wrote itself, where there were no nasty surprises at implementation, where it was 100% feature complete and bug-free, and you didn't have to patch it up a bunch of times. Maybe I'm squarely in the land of Santa Claus and the Easter Bunny, but I believe, deep down, all developers want to write that perfect piece of code. Unfortunately, real life has other ideas. Deadlines, unclear or conflicting requirements, ridiculous scope, being human – all these things keep us from the promised land of perfect code.

But here's the rub: though it may be satisfying to dream about, it's likely that you'll never produce truly perfect code for real-world applications. You'll sit down to write a piece of code, you'll do the best you can, taking into account everything you know about how the system works, how your piece of code fits into that system, and so forth. But we all know there will be mistakes – probably lots of them. And you'll do some testing, and the QA guys will do some testing, and the beta customers will do some testing, and then poof, the business-minded people in your organization will decide it's good enough to be released. At that point, the code isn't perfect, and every time you have to change that released code, you introduce risk into the system. Thousands or hundreds of thousands or millions of people are using it as is, and if you decide to make changes it might work differently for those people. This is risky, and the tools you use to help write your code must be cognizant of that fact.

Tools for Writing Imperfect Code

This article is about a certain kind of tool – static code analysis – that can be used to help you in writing good code. Not perfect code, but good code. As introductory computer science classes increasingly move to Java (even the high school AP computer science curriculum is Java-based now), the tools available to C/C++ developers should move over to Java as well. Over the last decade, as Java exploded in popularity, there have been tremendous breakthroughs in the area of practical static code analysis for defect detection. Today many commercial tools are available to do static code analysis of your C, C++, and Java code. I work for one such tool provider and I'll discuss our experience expanding from C/C++ into Java here. We'll explore how some of the concepts we used to analyze C/C++ code translated into the Java realm and the lessons we've learned in making this type of technology practical to help you write good code. First, I'll dig into some discussion of architecture and then I'll give you my philosophy on finding bugs automatically and the true purpose of these tools.



C++ and Java: What's the Same?

From a code analysis perspective, C++ and Java have a lot in common. Both require you to build some representation of the code into the guts of your analysis for dataflow analysis. This means breaking each function or method into basic blocks, computing a control flow graph, and having an analysis engine that can push checks down each possible execution path in the methods while keeping track of the relevant variables and their values. With this, each check can then pull out relevant constructs while analyzing the code. For example, if I'm looking for NULL exception problems, my NULL checker simply looks for places where objects are compared against NULL or assigned a NULL value, and then lets the analysis push down a path until I see a dereference when that value is NULL.

Listing 1 shows an example from the Struts framework. Notice that on line 171, the developer compares body against null. Unfortunately, the developer probably meant to make that comparison == instead of !=. In the case where the pointer is null, the code will skip over the assignment on line 172 and dereference the body variable on line 175. Oops. Listing 2 shows you what that code looks like in the interface of a static code analysis tool. The analysis engine pushes the checker down all the paths in this function. The checker notices the comparison against null, keeps track of the body value as being null when the condition on line 171 is false, and then reports a problem when it's derferenced as null. Simple enough, right?

False Positives and Java

Well, almost right. The biggest problem that the designer of any static code analysis tool faces is false positives. What is a false positive? Basically, any time the analysis reports a defect where there is none, that's a false positive. Some people call this noise, but I like to stay away from that term. Noise is a problem, but it's a different problem. To better understand a case that might trip up a static code analysis tool, take a look at Listing 3. The struts code from the previous example has been slightly modified to introduce a data dependence between the value of body and the value of body_tracker. Notice that after the test of body against NULL, the value of body_tracker will be 5 if body is not NULL and 12 if body is NULL. As such, there's no longer a NULL dereference on line 177 because it's guarded by the check of body tracker. This example is simple enough, but may fool some simple analysis engines into reporting the defect where there really is no problem at all because there's no possible execution path that leads body to be dereferenced when NULL.

False positives cause developers to lose trust in a tool. Why? Because the tool is wrong, and if it's wrong more often than it's right, eventually the user won't trust the tool at all. Fortunately, the techniques available for reducing false positives in C/C++ analysis translate rather nicely into the Java space. We simply provide additional checkers to search for "false paths" through the code – paths that can never

Ben Chelf is chief technology officer of Coverity. Before he co-founded Coverity, he was a founding member of the Stanford Computer Science Laboratory team that architected and developed Coverity's underlying technology. In his role at Coverity, Ben works with organizations such as the U.S. Department of Homeland Security, Cisco, Symantec, and IBM to improve the security and quality of software globally. He holds an M.S. and B.S. in computer science from Stanford University. Ben frequently provides expert insight into software security and quality to the press, public audiences, and in published writings.

be executed when the program is running. These additional checkers keep track of data flow in different ways, and any time they find a path that can't be executed, it's pruned from the analysis. This "false path pruning" is a key way to significantly reduce the false positive rate.

C++ and Java – What's Different?

There are a few key differences in analyzing C/C++ code versus Java code. Unlike C/C++, Java affords us more luxury in choosing which *code* to analyze. We chose to analyze bytecode instead of source code. There are tremendous advantages to looking for defects at the bytecode level. The biggest, of course, is the fact that the code has already been compiled – you don't have to deal with compiling the code and juggling the many different flavors of build systems out there. The disadvantage (if you can call it that) of analyzing bytecode instead of source code is that you need some way to tie the errors you find back to the source code. This means that the bytecode needs to have debugging symbols in it or the errors you produce won't be of much help in actually fixing the code.

The types of defects that you look for are also different. Defects in Java code have different runtime implications than their C/C++ counterparts. A NULL pointer dereference throws an exception in Java and crashes your system in C/C++. A resource leak in C/C++ happens any time heap-allocated memory isn't freed, but in Java, resource leaks occur under different circumstances – when clean-up must be done on an object that the garbage collector can't be responsible for.

Interprocedural Analysis

One key feature of the most powerful static code analysis solutions is their ability to understand what happens when one method calls another. This not only helps in finding more complex defects in the code, it also reduces the false positive rate because analysis mistakes are less likely. However, the analysis of Java introduces a challenge in this regard because virtually *every* method call is, er – virtual. This means that it's not so clear which instruction a virtual method call will jump to when the code is being analyzed. It depends on the runtime type of the object invoking the method. While this is a problem in C++ as well, it tends to be less systemic due to the fact that most people developing C++ code (a) don't always use objects in their code and (b) don't make all their methods virtual. To tackle this problem with a practical code analysis tool, we've developed techniques to infer the correct types of objects at runtime to determine which virtual methods could be instantiated at any given call site. Of course, our technology must make the appropriate trade-offs to retain as much precision as possible while still scaling to analyze large real Java systems. There's some great research out there to discuss the academic techniques from which we draw our ideas for implementing this in the real world. If you're interested in learning more, check Google for "Rapid Type Analysis" or "Class Hierarchy Analysis."

Noise

As I mentioned earlier, false positives are the number one challenge for static analysis. The number two challenge, and unfortunately a harder problem to deal with, is *noise*. How is noise different from a false positive? Noise is any issue reported by the analysis that, while technically correct from an analysis perspective, is something you just don't care about. It's obvious why this is so hard – it's completely subjective! Yet it's very important to address this to produce useful results. Take a look at Listing 4. Notice that on line 173 there's an extra space before the statement. Your static code analysis tool could report that extra space as a defect, but I'm willing to bet that most of us would consider that noise. The analysis isn't wrong per se – the statements don't line up – but I just don't care. Sure, this example is extreme, but there are less extreme cases that can be equally frustrating – even within checkers for things like NULL pointer exceptions. I've heard developers say, "Sure, but if that happens, we're totally hosed anyway, so it doesn't matter that it throws an exception there!" So the analysis can be spot on, producing an actual "defect" that could occur, but it's still reporting noise.

What To Look For

There's no silver bullet for eliminating noise, and there will always be a trade-off between the aggressiveness of an analysis and its false positive rate. But this brings me back to my initial point about the risk of changing your code. The purpose of a static code analysis tool, whether for C/C++ or for Java, is to help you find defects that would hurt the most, and to find them earlier in the software process. The purpose of these tools is *not* to find everything that's bad in your code, and that's a subtle distinction. There's too much risk associated with changing your code to address every little nitpick a static analysis tool can report. So when you're looking to add this type of technology to your arsenal of tools to help you ward off the bugs, take a close look at what it's going after. More "bugs" aren't necessarily better. Your time is valuable, and you don't want to waste it poring through false positive-ridden and noisy reports. Fortunately, there are tools out there that are on your side.

Listing 1
/org/apache/struts/taglib/bean/DefineTag.java

```
171                     if (body != null) {
172                         body = body.trim();
173                     }
174
175                     if (body.length() < 1) {
176                         body = null;
177                     }
```

Listing 2
/org/apache/struts/taglib/bean/DefineTag.java

```
Event branch_null: this.body is null
At conditional (2): this.body != null taking false path
171                     if (body != null) {
172                         body = body.trim();
173                     }
174
Event deref_while_null: this.body dereferenced while null
175                     if (body.length() < 1) {
176                         body = null;
177                     }
178                 }
```

Listing 3

```
170                 body_tracker = 5;
171                   if (body != null) {
172                     body = body.trim();
173                 } else {
174                           body_tracker = 12;
175                   }
176
177                 if (body_tracker != 12 && body.length() < 1) {
178                     body = null;
179                 }
```

Listing 4

```
170
171                   if (body != null) {
172                     body = body.trim();
173                            body_tracker = 12;
174                   }
175
```

EJB 3 Transactions

Understanding and using transactions with EJB 3

by Raghu R. Kodali
and Jonathan Wetherbee

Much of the work surrounding the design and development of enterprise applications involves decisions about how to coordinate the flow of persistent data. This includes when and where to cache data, when to apply it to a persistent store (typically the database), how to resolve simultaneous attempts to access the same data and how to resolve errors that might occur while data in the database is in an inconsistent state. A reliable database is capable of handling these issues at a low level in the database tier, but these same issues can exist in the middle (application server) and client tiers as well, and typically require special application logic. One of the principal benefits of using EJB 3 is its support for enterprise-wide services like transaction management and security control. In this article, we will explore how EJB 3 offers transaction services and how you can leverage them to meet your specific requirements.

Raghu R. Kodali is consulting product manager and SOA evangelist for Oracle Application Server. He leads next-generation SOA initiatives and J2EE feature sets for Oracle Application Server, with particular expertise in EJB, J2EE deployment, Web services, and BPEL. He holds a Masters degree in Computer Science and is a frequent speaker at technology conferences. Raghu is also a technical committee member for the OASIS SOA Blueprints specification, and a board member of Web Services SIG in OAUG. He maintains an active blog at Loosely Coupled Corner (www.jroller.com/page/raghukodali).

raghu.kodali@oracle.com

Understanding Transactions

A transaction is a group of operations that must be performed as a unit. These operations can be synchronous or asynchronous, and can involve persisting data objects, sending mail, validating credit cards, etc. A classic example is a banking transfer, in which one operation debits funds from one account (i.e., updates a record in a database table) and another operation credits those same funds to another account (updates another row in that same, or a different database table). From the perspective of an external application querying both accounts, there must never be a time when these funds can be seen in both accounts. Nor can a moment exist when the funds can be seen in neither account. Only when both operations in this transaction have been successfully performed can the changes be visible from another application context. A group of operations that must be performed together in this way as a unit is known as a transaction.

When the operations in a transaction are performed across databases or other resources that reside on separate computers or processes, this is known as a distributed transaction. Such enterprise-wide transactions require special coordination between the resources involved and can be extremely difficult to program reliably. This is where Java Transaction API (JTA) comes in, providing the interface that resources can implement and to which they can bind, in order to participate in a distributed transaction. The EJB container is a transaction manager that supports JTA and so can participate in distributed transactions involving other EJB containers, as well as third-party JTA resources like many database management systems (DBMS).

The ACID Properties of a Transaction

Transactions come in all shapes and sizes and can involve synchronous and asynchronous operations, but they all have some core features in common, known as their ACID components. ACID refers to the four characteristics that define a robust and reliable transaction: atomicity, consistency, isolation, and durability. Table 1 describes these four components.

EJB 3 addresses these requirements by providing a robust JTA transaction manager and a declarative metadata API that can be specified on interoperable, portable business components. Virtually all Java EE applications require transaction services and EJB brings them to the application developer in a very slick package. From its inception, the EJB framework has provided a convenient way to manage transactions and access control by letting the developer define the behavior declaratively on a method-by-method basis. Beyond these container-provided services, EJB 3 allows developers to turn control over to the application to define transaction event boundaries and other custom behavior.

EJB 3 Transaction Services

The EJB 3 transaction model is built on this JTA model, in which session beans or other application clients provide the transactional context in which enterprise services are performed as a logical unit of work. Enterprise services in the Java EE environment include the creation, retrieval, updating and deletion of entities; the sending of JMS messages to the queue; the execution of MDBs; the firing of mail requests; the invocation of web services; and JDBC operations.

EJB 3 provides a built-in JTA transaction manager, but the real power lies in the declarative services EJB offers to bean providers. Using metadata tags instead of programmatic logic, bean providers can seamlessly participate in JTA transactions and declaratively control the transactional behavior of each business method on an enterprise bean. EJB 3 extends this programming model by providing explicit support for both JTA transactions and non-JTA (resource-local) transactions. Resource-local transactions are restricted to a single resource manager, such as a database connection, but may result in a performance optimization by avoiding the overhead of a distributed transaction monitor. In addition,

application builders may leverage the container-provided (JTA-based) services for automatically managing transactions, or they may choose to take control of the transaction boundaries and handle the transaction begin, commit and rollback events explicitly. Within a single application, both approaches may be used alone or in combination if desired. Whereas the choice of whether to have the container or the application itself demarcate transactions is defined on the enterprise bean, the decision of which type of transaction model to use – JTA or resource-local – is determined when a given EntityManager is obtained from within an application. The persistent objects in the game – the entities – are entirely, and happily, unaware of their governing transaction framework. The transactional context in which an entity operates is not part of its definition, so the same entity class may be used in whatever transactional context the application chooses, provided an appropriate EntityManager is created to service the entity's life cycle events.

The EJB 3 container offers declarative demarcation of transaction events, along with the option to demarcate transaction events explicitly in the bean or in the application client code. Let's consider these two approaches separately, beginning with the default option: leveraging container-managed transaction (CMT) demarcation using declarative markup.

Container-Managed Transaction (CMT) Demarcation

EJB 3 provides built-in transaction management services that are available by default to session beans and MDBs. The container demarcates transaction boundaries and automatically begins and commits transactions based on declarative metadata provided by the bean developer.

When an EJB declares its transactional behavior in metadata, the container interposes on calls to the enterprise bean's methods and applies transactional behavior at the session bean's method boundaries. One of a fixed set of options may be specified for each method. The default behavior provided by the container is to check, immediately before invoking the method, whether a transaction context is associated with the current thread. If no transaction context is available, the container begins a new transaction before calling the method. If a transaction is available, the container allows that transaction to be propagated to the method call and made available to the method code. Then, upon returning from the method invocation, the container checks again. If the container was responsible for creating a new transaction context, it automatically commits that transaction after the method is exited (or, if an exception is thrown by that method, it rolls back the transaction it began). If it did not create the transaction, then it allows the transaction to continue unaffected. By interposing on the bean's method calls, the EJB container is able to apply transactional behavior at run time that was specified declaratively at development time.

Feature	Description
Atomicity	A transaction is composed of one or more operations that are performed as a group, known as a unit of work. Atomicity ensures that at the conclusion of the transaction, these operations are either all performed successfully (a successful commit), or none of them are performed at all (a successful rollback).
Consistency	A consistent transaction has data integrity. Consistency ensures that at the conclusion of the transaction, the data is left in a consistent state, so that database constraints or logical validation rules are not left in violation.
Isolation	Transaction isolation specifies that the outside world is not able to see the intermediate state of a transaction. Outside programs viewing the data objects involved in a transaction must not see the modified data objects until after the transaction has been committed.
Durability	The changes that result when a transaction is committed must become visible to the other applications.

Table 1 The ACID Properties of a Transaction

The default behavior described above is specified by the REQUIRED transaction attribute. You can attribute any one of the six demarcation options shown in Table 2 to any method on a session bean.

All six attributes are typically available for session bean methods, though certain attributes are not available on a session timeout callback method, or when the session bean implements javax.ejb.SessionSynchronization. MDBs support only the REQUIRED and NOT_SUPPORTED attributes. Here is an example of how you would specify the transaction behavior on a session bean method to override the transaction behavior specified (or defaulted) at the bean level:

```
@TransactionAttribute(TransactionAttributeType.SUPPORTS)
public CustomerOrder createCustomerOrderUsingSupports(Customer
customer)
throws Exception { ... }
```

Table 3 illustrates an EJB's transactional behavior, dependent on its transaction attribute and the presence or absence of a transactional context at the time the session method is called.

Bean-Managed Transaction (BMT) Demarcation

For some enterprise beans, the declarative CMT services may not provide the demarcation granularity they require. For instance, a client may wish to call multiple methods on a session bean without having each method commit its work upon completion. In this case, the client has two options: it can either instantiate its own JTA (or resource-local) transaction, or it can ask the session bean to expose transaction demarcation methods that the client can call to control the transaction boundaries itself.

To address this latter requirement, EJB offers enterprise beans a convenient way to handle their demarcation of transaction events. To turn off the automatic CMT demarcation services, enterprise beans simply specify the **@TransactionManagement(TransactionManagementType.BEAN)** annotation or assign the equivalent metadata to the session bean in the ejb-jar.xml file. With BMT demarcation, the EJB container still provides the transaction support to the bean. The primary difference is that the bean makes explicit calls to begin, commit and roll back transactions instead of using CMT attributes to declaratively assign transactional behavior to its methods. Also, the container does not propagate transactions begun by a client to beans that elect to demarcate their own transactions. While any given enterprise bean must choose one plan or the other (CMT vs. BMT demarcation) for its methods, both types of beans may interact with each other within a single transaction context.

In the last part of this article, we discuss JPA entity transaction behavior.

Jonathan Wetherbee is a consulting engineer and tech lead for EJB development tools on Oracle's JDeveloper IDE. He has over 12 years of experience in development at Oracle, having built a variety of O/R mapping tools and holding responsibility for Oracle's core EJB toolset since EJB 1.1. In 1999, he received a patent for his work on integrating relational databases in an object-oriented environment.

Jon is co-author of *Beginning EJB 3 Application Development: From Novice to Professional* (Apress, 2006), and enjoys speaking at user groups on EJB and related topics. Jon holds a BS in cognitive science from Brown University.

jon.wetherbee@oracle.com

How Entities Become Associated with a Transaction Context

From the preceding discussion about how the EJB server acts as a transaction coordinator in associating resources with a transaction context, you may have realized that a JPA entity's persistence context is the resource that gets associated with a transaction. In this way, a persistence context is propagated through method calls so entities in a persistence unit can see each other's intermediate state, through their common persistence context, if they are associated with the same transaction context. Also, the restriction that only one persistence context for any given persistence unit must be associated with a given transaction context ensures that for any entity of type T with identity I, its state will be represented by only one persistence context within any transaction context. Within an application thread, only one transaction context is available at any moment, but the EJB server is free to dissociate one persistence context from that thread and associate a new persistence context for the same persistence unit to satisfy transaction isolation boundaries. When the EJB server does this, the newly instantiated persistence context is not able to see the intermediate changes made to any entities associated with the suspended persistence context.

Transaction Attribute	Behavior
REQUIRED	This is the default transaction attribute value. Upon entering the method, the container interposes to create a new transaction context if one is not already available. If the container created a transaction upon entering the method, it commits that transaction when the method call completes.
MANDATORY	A transaction must be in effect at the time the method is called.
REQUIRES_NEW	The container always creates a new transaction before executing a method thusly marked.
SUPPORTS	This option is basically a no-op, resulting in no additional work by the container. If a transaction context is available it is used by the method. If no transaction context is available, then the container invokes the method with no transaction context.
NOT_SUPPORTED	The container invokes the method with an unspecified transaction context.
NEVER	The method must not be invoked with a transaction context.

Table 2 Container Transaction Attribute Definitions

Transaction Attribute	Client's Transactions	Transaction Associated with Business Method	Transaction Associated with Resource Managers
MANDATORY	None T1	Error T1	N/A T1
NEVER	None T1	None T1	None T1
NOT_SUPPORTED	None T1	None T1	None T1
REQUIRED	None T1	T2 T1	T2 T1
REQUIRES_NEW	None T1	T2 T3	T2 T3
SUPPORTS	None T1	None T1	None T1

Table 3 Client and Bean Transaction States for Each of Six Transaction Attributes

Container-Managed vs. Application-Managed Persistence Context

The persistence services in EJB 3 let you opt out of container-managed entity persistence altogether and manage the transaction life cycles of your entities explicitly within your application code. When an EntityManager is injected (or looked up through JNDI), it comes in as a container-managed persistence context. The container automatically associates container-managed persistence contexts with any transaction that happens to be in context at the time that the EntityManager is injected. Should an application wish to control how or whether its persistence contexts are associated with transactions, it may obtain an **EntityManagerFactory** (again, through container injection or JNDI lookup) and explicitly create the EntityManager instances that represent their persistence contexts. An application-managed persistence context is used when the EntityManager is obtained through an **EntityManagerFactory**—a requirement when running outside the Java EE container.

Transaction-Scoped Persistence Context vs. Extended Persistence Context

When an EntityManager is created, you may specify whether the persistence context that it manages should be bound to the life of a transaction, or whether it should span the life of the EntityManager itself. A persistence context that is created when a transaction is created, and destroyed when the transaction ends, is known as a transaction-scoped persistence context. A persistence context that is created at the time it is injected into the bean (or bound through a JNDI lookup), and is not destroyed until the EntityManager instance is itself destroyed, is called an extended persistence context. Only stateful session beans may use extended persistence contexts. At the time an EntityManager instance is created, its persistence context type is defined, and it may not be changed during the EntityManager's lifetime. The default type is transaction-scoped; to inject an EntityManager by specifying an extended persistence context, you may specify the injection directive with the following:

```
@PersistenceContext(type = PersistenceContextType.EXTENDED)
private EnterpriseManager em;
```

or you may define a persistence-context-ref element in the XML descriptor.

Summary

In this article, we began with a discussion of the concepts essential to all transaction behavior, and we then explored both the built-in, declarative features offered by the EJB container, as well as options to bypass this support and coordinate transactions in application code. We concluded by describing the ways that JPA entities can interact with EJBs in a transactional environment.

Now that you are familiar with how to set up and use EJB 3 transactions, you may wish to explore the many related areas also introduced in the EJB 3 and JPA. For an examination of these features, with code samples, check out *Beginning EJB 3 Application Development: From Novice to Professional* (Apress, 2006).

Introducing JReport Live™

JReport® 8

Bring Your Reports to Life

JReport Live is the only solution to empower your Java application with operational business intelligence. Now your application can support both operational reporting and analytics – making users and developers happy.

Powerful Analytics Based on Dynamic Cube Technology

For the first time your users can enjoy ad hoc reporting right from the application, working with the most current operational data. Users can now slice-and-dice data, expand/collapse groups, pivot and drill up/down/across any report. The power of JReport Live is based on our Dynamic Cube Technology. JReport Dynamic Cubes are easy to define and are instantly created with no overhead. Information is always current and presented in the context of the application.

An Enterprise-class Foundation for Operational BI

JReport Live is built on our powerful operational reporting solution, delivering all of the benefits of scalability, performance and security of JReport 8. Plus, you can combine reports into report sets for better performance, easy scheduling and fast deployment. The pipeline mode allows users to start viewing reports before generating a full report. JReport 8 can integrate with any Java EE application and any security scheme. In addition, JReport runs reports on demand, on a schedule or by event triggers.

Build Complex Precise Reports Quickly and Easily

Only JReport 8 is not constrained by a rigid report layout, it lets you mix and match report components and control precisely how they are presented on the page. With multimedia objects, Web controls and Web forms to further enhance reports, your users will be happy to work with the highly functional, interactive JReport Live reports.

Download your version of JReport 8 with JReport Live or call 240-477-1000 today.

Bring your Reports to Life with JReport Live Server

For more information, visit www.jinfonet.com/live

JReport®
JINFONET SOFTWARE

Configuring WebLogic Server 9.x JDBC

Data source connections

by Deepak Vohra

WebLogic Server 9.x provides database connectivity with data sources. A data source is a pool of database connections from which a connection can be obtained. A data source can be configured separately or as a multi-datasource. A multi-data source is collection of data sources. A data source is configured with a JNDI binding. A DataSource object is obtained with a JNDI lookup. A Connection object can be obtained from a DataSource object with the getConnection() method. WebLogic Server provides an administration console to configure a data source. WebLogic Server 9.x includes Type 4 JDBC drivers from DataDirect for DB2, Informix, Microsoft SQL Server, Sybase, and Oracle databases. JDBC drivers for other databases can be incorporated in the server by including the JAR files for the JDBC drivers in the server classpath.

New JDBC features in WebLogic Server 9.0 include support for JDBC 3.0, multiple JNDI names for a data source, and support for a Logging Last Resource transaction option. SQL Statement Timeout has been added to the connection pool configuration. SecondsToTrustAnIdlePoolConnection and PinnedToThread connection pool properties, which we'll discuss below, have been added to improve data source performance. The connection request failover feature has been improved. Statistics collection has been added for the different connection parameters for performance diagnostics. New features have been added to WebLogic Type 4 JDBC drivers and there's new identity-based connection pooling. Transaction, diagnostic, and security tabs are now part of the administration console for configuring a data source.

Deepak Vohra is a Sun Certified Java 1.4 Programmer and a Web developer.

dvohra09@yahoo.com

Setting the Environment

Download WebLogic Server 9.1. To install the application server double-click on the server910_win32 application. Click on the Next button in the BEA Installer. Select Create a New BEA Home in the Choose BEA Home Directory frame and specify a directory in the BEA Home Directory field. Click on Next. In the Choose Install Type frame select Complete and click on the Next button. Select any optional tools if required in the Optional Tools frame and click on the Next button. In the Choose Product Directory frame select the default Product Installation Directory and click on the Next button. Click on Next in the Create shortcut locations frame. BEA WebLogic Server gets installed.

Download the JDBC driver for the database. We'll configure JDBC connectivity with the MySQL database. So install the MySQL database and download the JDBC driver Connector/J.

Extract the MySQL JDBC driver zip file to a directory. Add the MySQL JDBC driver JAR file mysql-connector-java-3.1.11-bin.jar file to the CLASSPATH variable of the <weblogic91>\samples\domains\wl_server\bin\startWebLogic script file. <weblogic91> is the directory in which WebLogic Server 9.1 is installed. Double-click on the <weblogic91>\samples\domains\wl_server\startWebLogic to start the WebLogic examples server.

Figure 1 JDBC>data sources

Figure 2 Creating a new data source

Creating a Data Source

A data source is a pool of JDBC connections from which a connection can be obtained with the getConnection() method of a DataSource object. In this section we'll create a data source in the administration console. Access the administration console with the URL http://localhost:7001/console. In the administration console select the node Services>JDBC>DataSources.

To create a new JDBC data source click on New in the Data Sources table (see Figure 1).

Specify a data source name in the Create a New JDBC Data Source frame and a JNDI name for the data source. A data source is bound on a JNDI tree with a JNDI name. Select a Database Type. We'll create a data source with the MySQL database.

Select MySQL as the Database Type. Select MySQL's driver (Type 4) as the database driver and click on Next (see Figure 2).

A data source can be configured with any of the commonly used databases. WebLogic provides Type 4 JDBC drivers from DataDirect for DB2, Informix, Oracle, SQL Server, and Sybase. The DataDirect drivers are pre-installed in the <weblogic9.1>/server/lib directory (see Figure 3). The different JDBC Type 4 drivers included with WebLogic Server are listed in Table 1.

Databases other than those for which a JDBC driver is included can also be selected. If a JDBC driver other than the WebLogic driver is selected add the driver zip/JAR file to the CLASSPATH variable in the startWebLogic script (see Figure 4).

In the Transaction Options frame the transaction attributes of the data source are specified. If a XA driver is selected global transactions are automatically supported with the two-phase commit transaction protocol. A global transaction, or a distributed transaction, is a transaction that involves two or more transactions over multiple (or single) resource managers (a RDBMS database is a resource manager). A global transaction is managed by a Transaction Manager using JTA. For a non-XA data source to support global transactions, check the Global Transactions checkbox. Select the protocol that supports global transactions. The different transaction protocols are listed in Table 2.

In the Transaction Options page click on Next (see Figure 5).

In the Connection Properties frame specify the Database Name, test for the default database instance. Specify Host Name as localhost, Port as 3306, and user name as root. A password isn't required for the root username. Click on the Next button (see Figure 6).

In the Test Database Connection frame the driver class name, connection URL, and username for the MySQL database are specified. Click on the Test Database Configuration button to test the connection with the database (see Figure 7).

A message gets displayed indicating if a connection has been established. If database doesn't get connected an error message gets displayed. Click on the Next button.

In the Select Targets frame select a server where the data source will be deployed. To deploy to the examples server, select examplesServer and click on the Finish button (see Figure 8).

A data source gets configured and added to the Data Sources table (see Figure 9).

Click on the Activate Changes button to make the data source available to applications in the server.

Configuring a Data Source

In this section the data source created in the last section will be configured. Select the data source to configure in the Data Sources table. Select the Configuration tab (selected by default). In the Configurations frame the data source JNDI name can be modified (see Figure 10).

The other configuration options for a data source are listed in Table 3.

To configure the connection pool associated with data source select the Connection Pool link. Initial capacity, maximum capacity, and capacity increment can be set in the connection pool configuration (see Figure 11).

Some of the connection pool settings are listed in Table 4.

The transaction protocol settings can be configured with the Transaction link. Monitoring statistics can be collected with the Diagnostics link (see Figure 12).

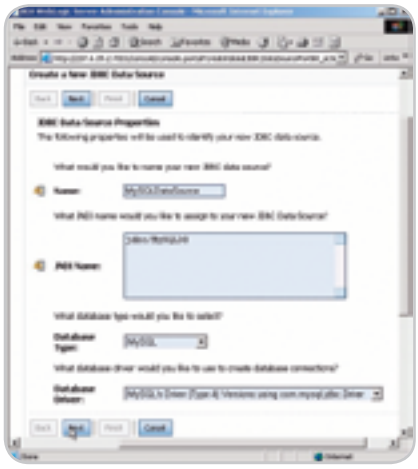


Figure 3 Specifying data source properties

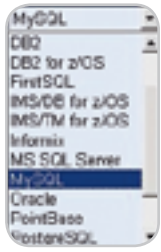


Figure 4 Databases supported

Figure 5 Selecting transaction options

Database	Versions Supported	Driver Classes	Connection URL
DB2	DB2 UDB 7.x, 8.1, and 8.2 on Linux, Unix, and Windows.	**XA** – weblogic.jdbcx.db2.DB2DataSource **Non-XA** – weblogic.jdbc.db2.DB2Driver	On Windows, Unix, Linux-jdbc:bea:db2://db2_server_name:port;DatabaseName=database
Informix	Informix 9.4 and later	**XA** – weblogic.jdbcx.informix.InformixDataSource **Non-XA** – weblogic.jdbc.informix.InformixDriver	jdbc:bea:informix://dbserver1:1543;informixServer=dbserver1;databaseName=dbname
MS SQL Server	MS SQL Server 7.0, SQL Server 2000(SP1, SP2, and SP3a)	**XA** – weblogic.jdbcx.sqlserver.SQLServerDataSource **Non-XA** – weblogic.jdbc.sqlserver.SQLServerDriver	jdbc:bea:sqlserver://dbserver:port
Oracle	Oracle 9i (R1 and R2), Oracle 10g (R1)	**XA** – weblogic.jdbcx.oracle.OracleDataSource **Non-XA** – weblogic.jdbc.oracle.OracleDriver	jdbc:bea:oracle://dbserver:port
Sybase	Sybase Adaptive Server 11.5, 11.9, 12.0, 12.5, 15.	**XA** – weblogic.jdbcx.sybase.SybaseDataSource **Non-XA** – weblogic.jdbc.sybase.SybaseDriver	jdbc:bea:sybase://dbserver:port

Table 1 WebLogic Type 4 JDBC Drivers

Transaction Protocol	% discount
Logging Last Resource (LLR)	LLR optimization provides better performance than a XA JDBC driver for insert, update, and delete operations. For read operations performance is better with an XA driver. Recommended over two-phase commit.
Two-Phase Commit	Emulates participation in a global transaction using JTA.
One-Phase Commit	The default setting. With one-phase commit only one resource can participate in the global transaction.

Table 2 Transaction Protocols

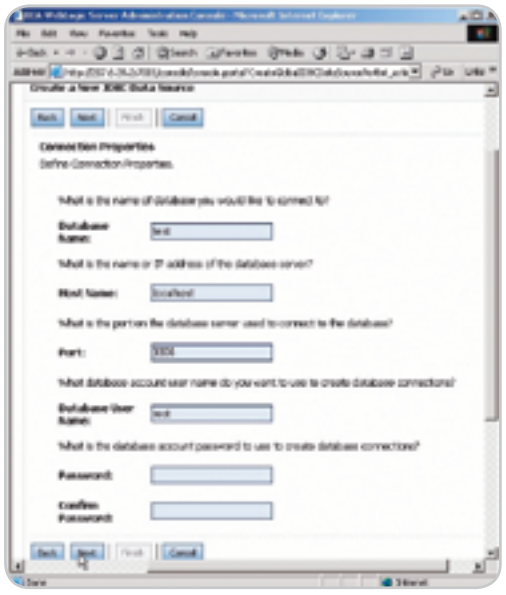


Figure 6 Specifying connection properties

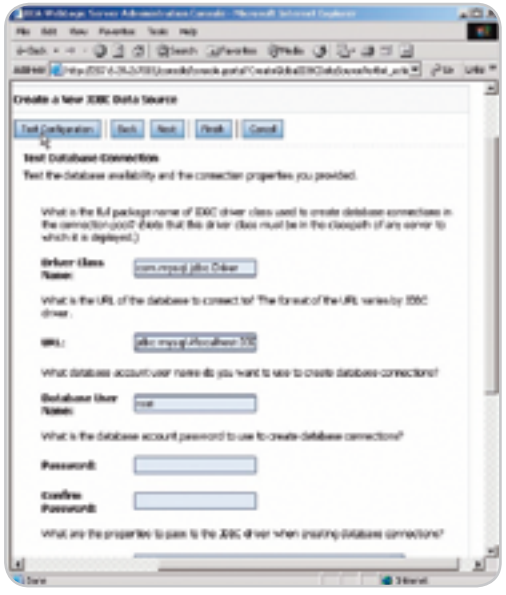


Figure 7 Testing the JDBC connection

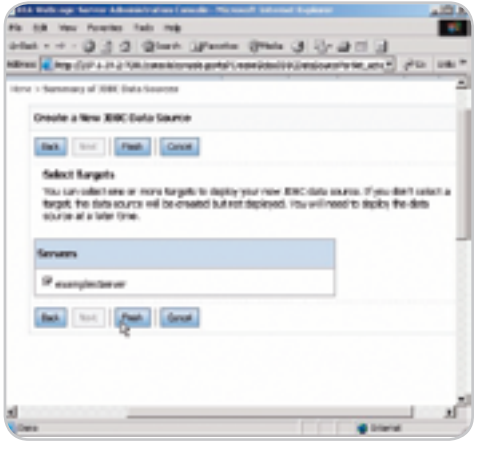


Figure 8 Deploying a data source on a server

Figure 9 New data source

Figure 10 Data source configuration

Figure 11 Setting connection pool attributes

Figure 12 Specifying profile collection

Data Source Setting	Description
Row Prefetch Enabled	For an external client, row pre-fetch fetches multiple rows from the server to the client in a single server access, improving performance.
Row Prefetch Size	If row pre-fetching is enabled, specifies the number of rows to fetch with row pre-fetching. Optimal size depends on the query.
Stream Chunk Size	Specifies the data chunk size for streaming data types.

Table 3 Data Source Configuration Options

Some of the data source profiles that can may be collected are listed in Table 5.

To monitor a data source select the Monitoring tab. To administer the WebLogic Server instances to which the data source is deployed select the Control tab. In a deployed server instance the statement cache can be cleared and server can be suspended or shut down.

Creating a Multi-Data Source

A multi-data source is an abstract group of data sources, which provides failover and load balancing around data sources. A multi-data source has a JNDI binding similar to a data source. To create a multi-data source click on the Services>JDBC>Multi Data Sources link in the administration console (see Figure 13).

In the Multi Data Sources table click on the New button to create a new multi-data source (see Figure 14).

In the Configure the Multi Data Source frame specify a data source name and a JNDI name for the data source. Select the algorithm type as Failover or Load Balancing. Click on the Next button (see Figure 15).

In the select Targets frame select the examplesServer or another server to deploy the multi-data source to. Click on the Next button (see Figure 16).

In the Select Data Source Type frame select XA driver for an XA data source or a non-XA driver for a non-XA data source. Click on the Next button (see Figure 17).

In the Add Data Sources frame add data sources from the Available list to the Chosen list. If new data sources are required click on the Create a New Data Source button. Then click on the Finish button (see Figure 18).

YOUR IDEA IS VISIONARY.
YOUR PLATFORM IS NOT.

MOBILITY. INTEROPERABILITY. INTELLIGENCE.

Meet the next-generation platform for distributed and edge computing– the only one that is completely platform independent.

VOYAGER Edge™

Your apps may be visionary, but they're dependent upon enterprise connectivity and distribution. When choosing a development platform, you need a solution that works everywhere- even where connectivity is limited. Voyager Edge™ is a proven, next-generation platform that uses intelligent mobile agent technology to solve mobility and performance issues in highly distributed environments. Faster than RMI, more reliable than client-server systems, Voyager agents can create ad-hoc networks that continue to work when they can't access the enterprise.

Voyager Edge gives architects maximum flexibility to freely develop dynamic, intelligent and decentralized applications in .NET and Java, on the devices and servers they need to target. These applications can run on centralized and edge devices, moving from one device to another while processing the same application. This unifying, next-generation platform can be used to gather, filter, analyze and distribute knowledge rapidly over today's increasingly heterogeneous wired and wireless networks.

Visit Recursion Software at www.recusionsw.com to make your vision a reality.

Download a free trial or intelligent mobile agent white paper at www.recursionsw.com.

Call 800.727.8674 to speak with an engineer.

RECURSION SOFTWARE, Inc.

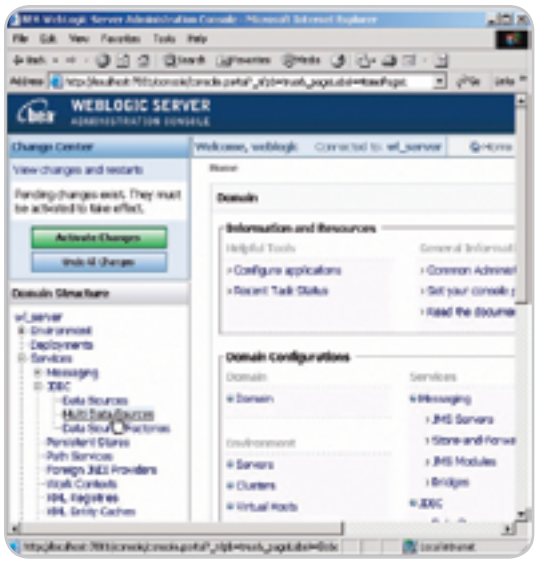


Figure 13 JDBC>multi data sources

Figure 14 Creating a new multi-data source

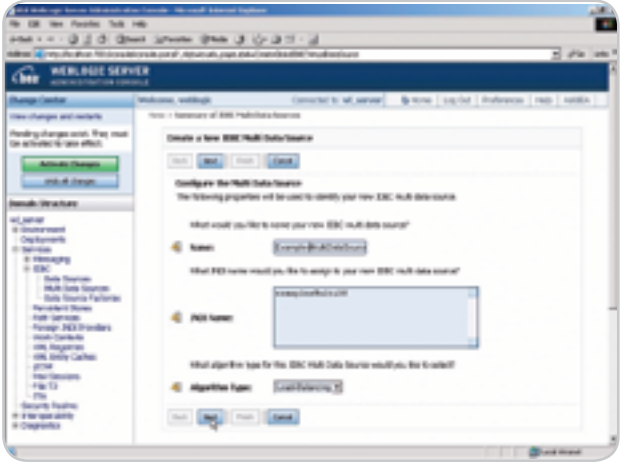


Figure 15 Specifying multi-data source attributes

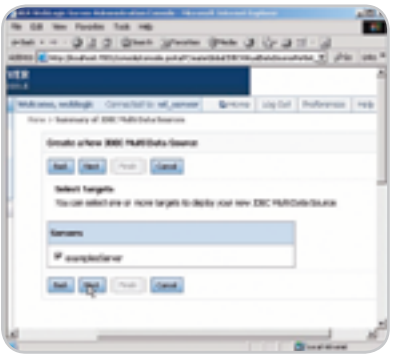


Figure 16 Deploying data source

Figure 17 Selecting data source type

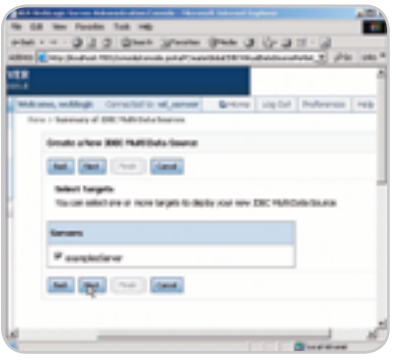


Figure 18 Adding data sources to a multi-data source

Figure 19 A new multi-data source

A new data source gets configured and added to the Multi Data Sources table. Click on the Activate Changes button to make the data source available to applications (see Figure 19).

A multi-data source can be configured by selecting the multi-data source link. The targets to which the multi-data source is deployed can be configured with the Targets tab. The data sources in the multi-data source can be configured with the Data Sources link in the Configuration tab. The multi-data source JNDI name can be modified in the Configuration>General frame. The Algorithm Type specifies the algorithm used to select a data source from which a connection is obtained. If the algorithm type is Failover, connection requests are sent successively to the data sources in the list until a connection is obtained or the end of the data source list is reached. If the algorithm type is a Load Balancing connection request the load is distributed evenly over the data sources in the list. If load balancing is the selected connection, failover is also provided with the connection requests being sent to different data sources in the list until a connection gets established or the end of the data source list is reached (see Figure 20).

A multi-data source also provides the Failover Request If Busy, Failover Callback Handler, and Test Frequency settings. For a multi-data source with the Failover algorithm if Failover Request If Busy is selected the connection request is sent to the next data source if all the connections in a data source are busy. The Failover Callback Handler specifies the application class to handle the callback sent when a multi-data source is ready to send a Failover connection request to another data source. Test Frequency specifies the interval in x number of seconds after which connections are tested. If a connection fails the connection is closed and reopened. If the connection fails again the connection is closed (see Figure 21).

Performance Tuning JDBC

Some of the JDBC design considerations to improve performance include the selection of the JDBC driver. The WebLogic Type 4 JDBC

drivers from DataDirect provide comparable performance. The connection pooling provided by WebLogic Server data sources improves performance by keeping a pool of connections available for JDBC applications. Connection don't have to be opened and closed for each client. Tune the number of connections by setting the Initial Capacity equal to Maximum capacity in a connection pool. Maximize the reuse of connections rather than closing and opening connections. A connection may be pinned to a thread by setting the PinnedToThread configuration pool attribute. Close connections after a connection is not required.

Test Connections on Reserve tests connections before making a connection available to a client, but connection testing can reduce performance. To prevent frequent connection testing set the connection pool attribute Seconds to Trust an Idle Pool Connection, which specifies the number of seconds for which a connection returned by a client isn't tested with a SQL query. Data source performance can be improved by selecting Row Prefetch Enabled and an optimal pre-fetch size in configuring a data source. Row pre-fetching improves performance by fetching multiple rows from the server to an external client. Caching statements improves performance by reusing statements rather than creating new ones. Statement caching is specified in the connection pool configuration.

Developing a JDBC JSP Application

In this section we'll develop a JSP application to retrieved data from a MySQL database table with the data source configured in the Creating a Data Source section. First, create an example database table in the MySQL database. Access the MySQL database with the command mysql.

```
>mysql
```

To create a database table in the test database instance login to the test database.

```
mysql>use test
```

Connection Pool Setting	Description
Initial Capacity	The initial number of connections in the connection pool. Also specifies the minimum number of available connections in the connection pool.
Maximum Capacity	The maximum number of connections in the connection pool.
Capacity Increment	The number of connections added to the connection pool when connections aren't available in the connection pool.
Statement Cache Type	Specifies the algorithm used to cache prepared statements. If the value is LRU, the least recently used statement in the cache is replaced when a new callable or prepared statement is created. If the value is FIXED, a fixed number of callable and prepared statements are cached.
Statement Cache Size	Specifies the number of prepared and callable statements in the cache. WebLogic Server performance increases by reusing statements in the cache rather than reloading statements. Each connection in the connection pool has a statement cache.
Test Connections on Reserve	If the Test Connections on the Reserve checkbox is selected the connections are tested before being given to the client. Test is required for connection pools in a multi-data source created with the Failover algorithm. If Test Connections on Reserve is selected Test Table Name should also be specified.
Test Frequency	Specifies the seconds interval to test unused connections in a connection pool. If the test fails the connection is closed and reopened. If the test fails again, the connection is closed. If Test Frequency more than 0 is specified Test Table Name should also be specified.
Test Table Name	The database table to use to test connections. To improve testing specify a table with few/no rows. The SQL query to test connection may be specified with: SQL <query>. <query> is the query to use to test the database connection.
Init SQL	Specifies the SQL statement to use to initialize a connection with a database. The SQL statement is specified with: SQL <sql statement>. <sql statement> is the SQL statement to use to test the connection. A database table canbe specified by not specifying 'SQL' at the start of the field. If a database table is specified a database connection is tested with the SQL statement "SELECT count(*) from InitSQL"
Shrink Frequency	Specifies the wait time for reducing the connection pool size to a pre-incremented value.
Connection Creation Retry Frequency	Specifies the number of seconds between attempts to establish a connection with a database.
Inactive Connection Timeout	Specifies the number of seconds after which an unused connection is returned to the connection pool.
Login Delay	Specifies the number of seconds to delay before establishing a connection with a database. Used for database servers that can't handle successive connection requests.
Maximum Waiting for Connection	Specifies the maximum number of connection requests waiting to obtain a connection from the connection pool.
Connection Reserve Timeout	Specifies the number of seconds after which a connection request will time out.
Statement Timeout	Specifies the number of seconds after which a statement will time out.

Table 4 Connection Pool Settings

Create a table, Catalog with the SQL script in Listing 1.

Create a JSP, catalog.jsp. In the JSP import the java.sql, javax.sql, java.util, and javax.naming packages.

```
<%@ page

import="java.sql.*,javax.sql.*,java.
util.*,javax.naming.*"

%>
```

Create a InitialContext object.

Figure 20 Configuring a multi-data source

Figure 21 Specifying multi-data source attributes

Figure 22 Output from the JDBC application

```
InitialContext ctx=new InitialContext();
```

Create a DataSource object from the JNDI name of the MySQL data source.

```
DataSource ds=(DataSource)ctx.lookup("jdbc/
MySQLDS");
```

Obtain a JDBC connection from the DataSource object.

```
Connection connection=ds.getConnection();
```

Create a Statement object from the Connection object.

```
Statement stmt=connection.createStatement();
```

Run a SQL query with the executeQuery() method to return a ResultSet object. Specify a SQL query that selects all of the columns in the example database table Catalog.

```
ResultSet resultSet=stmt.
executeQuery("Select * from
Catalog");
```

Create an HTML table with a column corresponding to each of the rows in the result set. Add a header row to the HTML table. Iterate over the result set and add row values to the HTML table.

The JSP, catalog.jsp, to generate a HTML table from the example database table with the data source configured in the WebLogic Server is in Listing 2.

To run the JSP in WebLogic Server copy the JSP to the <weblogic91>\samples\server\examples\build\mainWebApp directory. Invoke the JSP with the URL http://localhost:7001/catalog.jsp. The JSP runs in WebLogic Server and generates an HTML table (see Figure 22).

Conclusion

WebLogic Server 9.0/9.1 has a new feature, multi-data source, a group of data sources with a JNDI name binding. A multi-data source facilitates maximum data source availability. A separate connection pool configuration in WebLogic Server 8.1 has been removed. The data source configuration in WebLogic Server 9.x provides enhanced connection request failover and load balancing between data sources.

Listing 1: Catalog.sql

```
CREATE TABLE Catalog(CatalogId INTEGER
PRIMARY KEY, Journal VARCHAR(25), Publisher VARCHAR(25),
 Edition VARCHAR(25), Title Varchar(45), Author Varchar(25));

INSERT INTO Catalog VALUES('1', 'Oracle Magazine', 'Oracle
Publishing', 'Nov-Dec 2004',
'Database Resource Manager', 'Kimberly Floss');

INSERT INTO Catalog VALUES('2', 'Oracle Magazine', 'Oracle
Publishing', 'Nov-Dec 2004',
'From ADF UIX to JSF', 'Jonas Jacobi');

INSERT INTO Catalog VALUES('3', 'Oracle Magazine', 'Oracle
Publishing', 'March-April 2005',
'Starting with Oracle ADF ', 'Steve Muench');
```

Listing 2: Catalog.jsp

```
<%@ page contentType="text/html"%>
<%@ page

import="java.sql.*,javax.sql.*,java.util.*,javax.naming.*"

%>
<html>
  <head>
    <meta http-equiv="Content-Type" content="text/html">
    <title>JDBC JSP Application</title>
  </head>
<body>
 <%
    InitialContext ctx=new InitialContext();
    DataSource ds=(DataSource)ctx.lookup("jdbc/MySQLDS");
    Connection connection=ds.getConnection();
    Statement stmt=connection.createStatement();
    ResultSet resultSet=stmt.executeQuery("Select * from
Catalog");%>

    <table border="1" cellspacing="0">
        <tr>
          <th>CatalogId</th>
      <th>Journal</th>
      <th>Publisher</th>
         <th>Edition</th>

         <th>Title</th>
         <th>Author</th>
        </tr>
      <%
   while (resultSet.next())
        { %>
         <tr>


<td><%out.println(resultSet.getString(1));%></td>


<td><%out.println(resultSet.getString(2));%></td>


<td><%out.println(resultSet.getString(3));%></td>


<td><%out.println(resultSet.getString(4));%></td>


<td><%out.println(resultSet.getString(5));%></td>


<td><%out.println(resultSet.getString(6));%></td>
         </tr>
    <% } %>
    </table>
  </body
```

Enterprise AJAX is ICEfaces

- Create secure Rich Internet Applications (RIA)
- Develop in Java, not JavaScript
- Transform the User Experience

Visit www.ICEfaces.org to try it out!

RIA Solution for SOA

Rich User Experience

Create a new class of collaborative and dynamic enterprise applications. Unleash the unique power of Ajax Push Technology to deliver server-initiated, instantaneous presentation updates. Easily migrate existing JSP-based and traditional client-server applications to RIAs.

Standards-Based

Develop and deploy scalable RIAs in pure Java using an integrated Ajax framework complete with rich JSF-based components. Harness the power of Ajax and leverage the entire standards-based Java EE ecosystem of familiar tools and runtime environments.

Performance, Scale and Security

Deploy rich enterprise applications with advanced Ajax connection management for maximum application reliability. Seamlessly scale applications across clustered Java EE servers. Extend the existing web security model and avoid transferring sensitive business logic and data to the client browser.

ICESOFT
THE RICH WEB COMPANY

ICESOFT TECHNOLOGIES
Toll Free 1.877.263.3822
www.icesoft.com

Are Vendors Becoming More in Charge of Java Enterprise Edition...

...or is Sun losing control over Java EE?

by Andrei Iltchenko

One of the things that kept me and my team busy over the past couple of years was starting to support J2EE 1.4 in OptimalJ – a goal that we accomplished with the release of OptimalJ 4.2. Now that this job is complete, it's interesting to look back and ponder the experience and learn from it.

As OptimalJ is a vendor-neutral modeling tool that generates J2EE applications, we tend to avoid application server–specific features that are not mandated by J2EE. Typically our first take is to implement our code-generation modules by the letter of the spec and then ensure we fill in the necessary bits for a particular vendor by generating additional vendor-specific deployment descriptors. This task, quite feasible at first sight, proved rather intricate with J2EE 1.4. Why? One reason is incomplete implementations of certain parts of the spec. However, there were more causes as you'll see later. But first let's recap the history of J2EE 1.4.

The spec was released in November 2003. The first J2EE product certified by Sun was JBoss 4.0.0 released in July 2004. IBM's WebSphere 6.0 went out in October 2004, while BEA's WebLogic offering hit the market in July 2005. All these products passed Sun's J2EE 1.4 CTS. For an idea of what the J2EE CTS embodies, take a look at this interview: http://www.theserverside.com/tt/articles/article.tss?l=SunInterview. Here's a relevant quote from http://java.sun.com/javaee/overview/faq/j2ee.jsp (my emboldening):

J2EE technology is a set of standards that many vendors can implement. The vendors are free to compete on implementations but not on standards or APIs. Sun supplies a comprehensive J2EE Compatibility Test Suite (CTS) to J2EE licensees. The J2EE CTS helps ensure compatibility among the application vendors, which helps ensure portability for the applications and components written for the J2EE platform. ***The J2EE platform brings Write Once, Run Anywhere (WORA) to the server.***

Andrei Iltchenko is a development lead at Compuware Corporation where he works on the MDA product OptimalJ and is responsible for the business logic area of OptimalJ-generated J2EE applications. He is also a Sun certified Java developer for Java Web Services, a Sun Certified Business Component Developer, a Sun Certified Developer, and a Sun Certified Programmer.

andrei.iltchenko@nl.compuware.com

Problems Encountered

Our first surprise was with the new MDB (message-driven bean) contract. One vendor's product simply couldn't cope with the new activation configuration concept. This is really the core functionality of EJB 2.1 and JCA 1.5. In EJB 2.1, MDBs are decoupled from JMS and their component contract is changed to be extremely generic and extensible. They can receive messages from virtually any inbound resource adapter, not necessarily a JMS adapter. If you want to learn more about the interaction between resource adapters and MDBs in J2EE 1.4, take a look at this article: http://www.theserverside.com/tt/articles/article.tss?l=J2EE1_4. It is amazing how the product could pass Sun's J2EE 1.4 CTS without implementing this support.

The next wrinkle was with the timer service – a long-asked-for missing piece of J2EE that enables enterprise components to have a reliable and transactional timer. An enterprise component can request that a callback method be invoked on it after a given passage of time (possibly repeatedly). The component can query for pending callbacks, cancel some of them, add new ones, etc.

An important consideration here is to note that in EJB 2.1 all instances of the same stateless session bean (SLSB) and the same MDB are considered equal, i.e., if the container has created and pooled n instances of SLSB BeanA and one such bean instance has scheduled a time notification, the container can pick any of the *n* instances in the pool to service the request. While the timer notification is pending, any of the *n* BeanA instances that the container invokes can query for pending timer notifications and see that the notification is pending. That instance can cancel it, so that the timer event will be annulled and not invoked on any of the *n* instances.

Great was my surprise when I saw that one vendor implemented the timer service without regard for the above fact. The vendor's product sometimes allowed one instance of BeanA to observe the pending timer notification as cancelled and then suddenly again as pending – all in the same transaction!

A friend of mine, who works at another company, had a similar experience with his vendor (also certified by Sun as J2EE 1.4 compliant). He was deploying his application in a cluster and used the timer service with SLSBs. The phenomenon he witnessed was this: he had a pool of *n* BeanA instances deployed to multiple nodes in the cluster. When any of the *n* BeanA instances scheduled a timer notification, only the BeanA instances running on the same node as the bean instance that scheduled the notification

could see and service it. BeanA instances running on different nodes would neither see the pending notification nor be able to examine or cancel it, which essentially breaks the EJB requirement that all instances of the same SLSB have the same object identity.

With such surprises from certified J2EE 1.4 servers, how can you write a portable J2EE component that uses the timer service and enjoy the "Write Once, Run Anywhere" experience?

All this was small time compared to the issues we ran into when dealing with the highlight of J2EE 1.4 – support for Web services. The details of a compliant J2EE 1.4 Web services implementation are spelled out by three constituent specs that J2EE 1.4 includes by reference: *Web Services For J2EE, Version 1.1* (aka JSR-109/JSR-921), *JAX-RPC, Version 1.1*, and *WS-I Basic Profile, Version 1.0*. The last specification is crucial in ensuring interoperability of Web service components developed with J2EE.

One cardinal requirement of a compliant Web services stack is that it can handle literal serializations/deserializations of Java Data Models into/from XML. The need to support literal serialization of data calls for an XML Schema–aware Web service. If a vendor's Web service stack is not XML Schema aware, you will very likely have issues with the interoperability of the Web services you deploy to the vendor's product.

When we started testing our code generation modules on one vendor's product, we were in for a big surprise. The application server had a non-XML Schema–aware Web services stack, which effectively meant no proper support for WS-I Basic Profile. Why wasn't such a major omission caught by J2EE 1.4 CTS?

That wasn't all. Two products we tried couldn't handle overloaded methods in service endpoint interfaces (SEIs) of J2EE components. For an idea of what the SEI is, refer to my earlier article "Moving to SOA in J2EE 1.4" (http://java.sys-con.com/read/180362.htm).

Another major obstacle was with document-literal bindings. One of the three vendor products we support required that additional Java code artifacts called "wrapper beans" be generated and registered in a JAX-RPC type mapping DD for every operation of a SEI that is exposed as a Web service component with a document-literal binding, while two other products wouldn't work when such extra code artifacts were generated and registered. Yet more unfortunate was the fact that the product requiring the extra code artifacts left it up to the component developer to provide these rather than generate them automatically during deployment. Being quite unhappy with such a state of affairs, I decided to contact the "Web Services for Java EE" specification lead with a view to ensuring that the issue be resolved to the benefit of Java EE developers and the "Write Once, Run Anywhere" goal. One reply I got back from the spec lead was this:

We had clarified to you before on this issue that there is not much we can do for the JAX-RPC case with JSR-109 v1.1 or v1.2. There are J2EE 1.4 compliant application server vendors who have passed the CTS (Compatibility Test Suite) and changes proposed by you would mean that they would now need to fix their implementations to conform to this new requirement. This will not be received well by vendors who have all ready spent their resources getting J2EE 1.4 certification. While I do appreciate your intentions of improving developer experience, this may be a bit too disruptive for the application server vendors/licensees.

We have made improvements in the deployment/packaging process with the latest release of JSR-109 v1.2 (part of Java EE 5.0) for JAX-WS endpoints.

Unfortunately we found more inconsistencies with document-literal bindings. One J2EE 1.4 certified vendor couldn't deal with Web service components featuring methods without a return type when a component's binding was document-literal.

Another problematic area was the JavaBean-based nature of SOAP serialization rules (http://java.sys-con.com/read/180362.htm). When a J2EE Web service component receives or returns a value mapped to a JavaBean, the container is supposed to serialize it to/deserialize it from XML by using the *java.beans.Introspector* class to examine its properties. Two out of three products we tried didn't do that in the initial version of their products.

It's true that it is all but inevitable that a complex product has bugs, still most of the issues I presented above are of a bigger scale than simple bugs and should have been flagged with the J2EE 1.4 CTS.

Where Does Modeling Come In?

With the differences in implementing the same J2EE 1.4 features I described, you should probably start questioning how feasible it would be to maintain a J2EE 1.4 application that *needs to run on more than one application server* and not be rigged toward a given vendor. Based on the experience we have gone through, I've come to the conclusion that doing that might be quite a pain and will definitely not scale as the size of your application's code base increases. This, in my opinion, is a valid example in which model-driven architecture and development delivers the fruits so long as you have chosen a mature and reputable MDA product that supports J2EE.

You are welcome to discuss the topic on my blog: http://blogs.compuware.com/cs/blogs/andrei_iltchenko/default.aspx.

> "As OptimalJ is a vendor-neutral modeling tool that generates J2EE applications, we tend to avoid application server–specific features that are not mandated by J2EE"

What Is SCA?

A simple model for creating service-oriented applications

by Simon Laws, Haleh Mahbod, and Raymond Feng

Service Component Architecture (SCA) is a simple model for creating service-oriented applications. This article highlights the benefits of SCA and introduces SCA concepts by walking through an example. The example has been developed using the Apache Tuscany open source project (http://incubator.apache.org/tuscany/). All the sample code in this article is licensed under the Apache License 2.0 (http://www.apache.org/licenses/LICENSE-2.0) and the resources with the article gives a link to the sample files. Both the Apache Tuscany and PHP SCA_SDO (http://pecl.php.net/package/sca_sdo) projects provide a free service oriented infrastructure for creating, packaging, deploying, and managing applications built with the SCA programming model.

The SCA programming model itself is described by a set of specifications that are being developed by many vendors and individuals contributing to the Open Service Oriented Architecture collaboration (http://www.osoa.org).

SCA

Service Oriented Architecture (SOA) is an architectural approach driven by the need to overcome the challenges of tightly coupled and department-specific applications. SOA promises benefits such as improved business agility, improved flexibility, cost reduction, and the easy sharing of information in heterogeneous and distributed environments.

SOA provides a blueprint but implementing an SOA remains a challenge. The choice of technology available to the implementer is bewildering and skills in a variety of technologies are required to be successful. Service Component Architecture (SCA) addresses the complexity of developing an SOA solution through its simple model for creating service-oriented applications for the whole enterprise – from the client to the back-end. Businesses using SCA can benefit from the following:

- ***Rapid development and increase in productivity***: SCA views an application as a set of connected components. It provides a simple language-neutral component model for implementing new components or reusing existing components. A component can be implemented in any language supported by an SCA runtime. SCA promotes true loose coupling by separating component implementation from the details of component composition. This bottom-up development style allows the developer to focus on developing business-related code without worrying about how this will fit into the overall solution.
- ***Higher organizational agility and flexibility***: SCA also supports a top-down development approach of creating business solutions with its flexible service assembly model. SCA components can be wired together in a composition. A component can be replaced with another component in the composition as long as they share the same contract. The composition can be adjusted to IT infrastructure requirements such as service connections, transport protocols, transactions, security, and reliable messaging. Selectable transport bindings make solutions available in the widest possible set of deployment situations.
- ***Return on Investment through reuse***: The SCA component model makes it very easy to leverage investments made in existing applications and services. Its standardized approach to encapsulation and interface abstraction enables service reuse through wiring and rewiring to construct new applications. SCA itself is technology-neutral and isn't intended to replace existing technology. It simply provides a component composition model that describes how new and existing services are assembled.

Figure 1 is taken from the SCA Assembly Model specification (http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications) and shows the main artifacts of SCA.

The dark blue boxes (Component A and Component B) show components. Components are at the heart of SCA as they encapsulate business logic. Depending on runtime support, components implemented using any programming technique can be included. For example, Apache Tuscany currently supports the Java language, JavaScript, Ruby, Python, and C++ component types and provides an extension API for building new extensions.

Simon Laws is with IBM and is working with the open source Apache and PHP communities to build Java, C++ and PHP implementations of the Service Component Architecture (SCA) and Service Data Object (SDO) specifications.

simon_laws@uk.ibm.com

SCA components can have properties (the yellow boxes shown at the top of components A and B). Properties control the behavior of the component and can be changed at deployment time. For example, a stock quote application might have a property that indicates the currency that stock values will be quoted in.

SCA components describe the interfaces that they expose for other components to call, shown as the green arrows on the left-hand side of the component boxes and called "services" in SCA. Components also describe the interfaces of other components that they expect to call as the business logic executes, shown as the pink arrows on the right-hand side of the component boxes and called "references" in SCA. These exposed services and references can be "wired" together to describe a working system.

The diagram shows two components, A and B, assembled together within the bounds of a larger "composite," called composite A. The SCA composite describes a collection of wired components and, as you can see, the composite also echoes those services and references that must be exposed beyond the bounds of the composite. Wiring together components within a composite is akin to building a tightly coupled application that may run in a single process. Wiring together the services and references exposed by a composite represents a more loosely coupled system where each composite may run in a separate process or processor and is connected over a network with various protocol/transport bindings. This way SCA provides a consistent model for describing standalone and distributed applications.

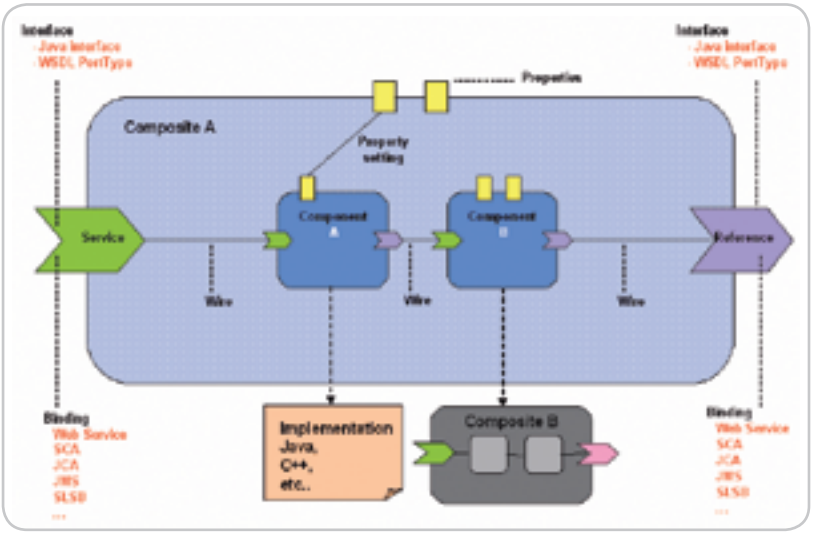


Figure 1 The artifacts of SCA

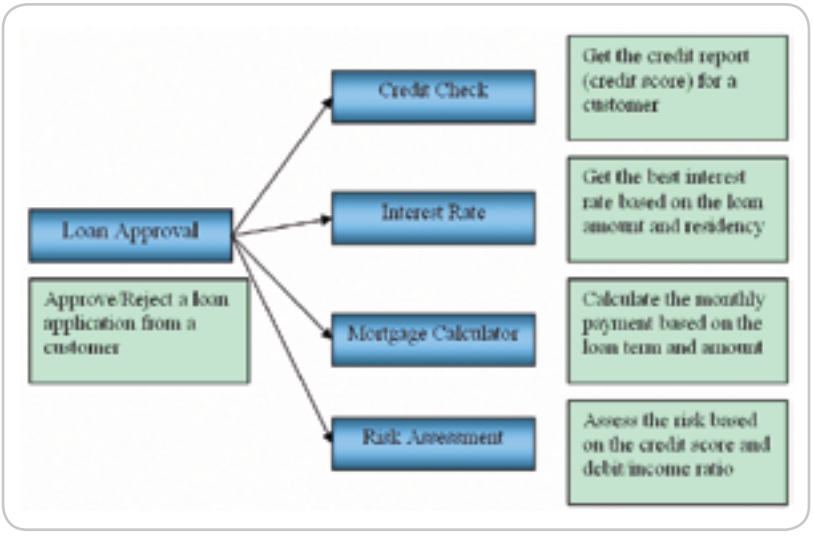


Figure 2 The loan approval application

An Example Scenario

We'll use the fictional MostMortgage company's mortgage loan approval application to introduce SCA in more detail. The loan approval application accepts a mortgage request including the customer's details and the requested loan amount. It first checks the customer's credit to make sure the credit score meets the minimum requirement. The interest rate is determined based on the principal requested, the term of the loan, and the customer's home state. It then uses a mortgage calculator to calculate the ratio by dividing the potential monthly payment by the customer's income. The ratio and credit score are passed to do a risk assessment that makes the final decision (see Figure 2).

Using SCA To Implement the Mortgage Loan Approval Application

In the next sections we'll implement the loan approval application using SCA and walk through the creation of individual SCA artifacts. At a high level the loan approval application can be broken down into a number of SCA components that are assembled together into a composite. The components in this composite consist of Loan Approval, Credit Check, Interest Rate, Mortgage Calculator, and Risk Assessment components. The entire composite is deployed in a SCA system (see Figure 3).

SCA Components

An SCA component is the basic building block for creating SOA applications and is characterized by three distinct and yet related pieces of information: a) The program logic that provides the function of the building block (referred to as implementation), b) The definition of how this building block might interact with other components (referred to as component type) c) The concrete description of how this building block fits with all the other blocks to build a solution (referred to as assembly or composition). We'll explain each in more detail in the following sections and give examples but here's an overview.

- ***Component Type***: The component implementation is provided using any programming language that's supported by an SCA runtime. Component implementers are free to write in any style they're comfortable with but are bound by the services, references and properties, as defined by the component type, in the way that they interact with other SCA components. The SCA specifications describe how each programming language maps to SCA.
- ***Component Type***: Component type describes the shape of a component in terms of the services it exposes, the references it depends on and the properties that control the component's behavior. Component-type information can be found either in a file where, by convention, the name is ImplementationFileName.componentType and/or by introspection of the component implementation.

Haleh Mahbod is a program director with IBM, managing the team contributing to the Apache Tuscany as well as SOA for PHP open source. She has extensive development experience with database technologies and integration servers.

mahbod@us.ibm.com

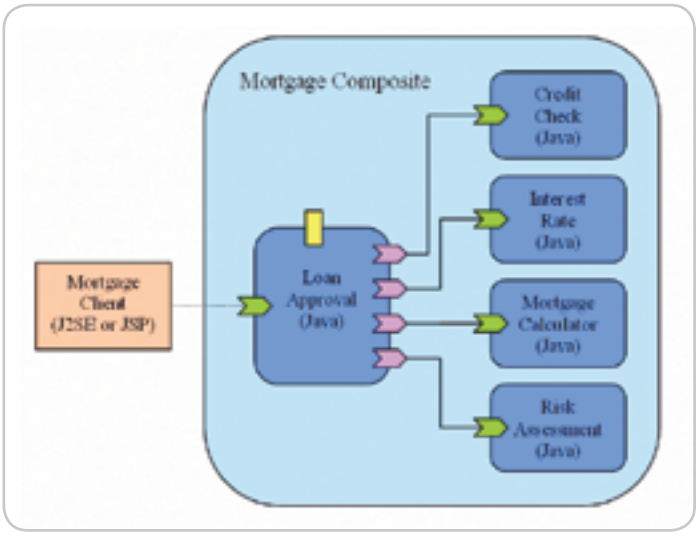


Figure 3 The loan approval application presented as a composition of SCA components

- ***Component Composition/Assembly***: Once a component's implementation and its component type are defined it's ready to be assembled into a network of services that together provide an SOA solution. The assembly is defined in an SCA composite file. The SCA runtime uses the information in this file to instantiate an SCA application.

SCA defines an XML format called Service Component Description Language (SCDL). SCDL is the XML format of component-type files and composite files. For example, the loan approval application's MortgageCalculator component has both component-type and component-implementation files and the MortgageCalculator component is described and wired together with other components in a composite file (see Figure 4).

Component Type

The MortgageCalculator component-type file (MortgageCalculator.componentType) describes the single service that components of this type provide. The MortgageCalculator component doesn't reference other components and doesn't provide any settable properties so <reference> and <property> elements don't appear.

```
<componentType>
    <service name="MortgageCalculatorService">
        <interface.java interface="mortgage.MortgageCalculator"/>
    </service>
</componentType>
```

Component Implementation

The class "mortgage.MortgageCalculatorImpl" (MortgageCalculatorImpl.java) contains the business logic for this component.

```
public class MortgageCalculatorImpl implements MortgageCalculator {
    public double getMonthlyPayment(double principal, int years,
float interestRate) {
        double monthlyRate = interestRate / 12.0 / 100.0;
        double p = Math.pow(1 + monthlyRate, years * 12);
        double q = p / (p - 1);
        double monthlyPayment = principal * monthlyRate * q;
        return monthlyPayment;
    }
}
```

In the next section (Component Services) we show that if we chose to use annotations, as we can in the Java language, the component-type information can be included in the implementation file. Most of the code snippets in this paper use annotations to provide component-type information instead of using a component-type file.

Component Services

Let's take a look at how a component offers a service to others. In the following example MortgageCalculator exposes a service that contains one method, called getMonthlyPayment, by using a @Service annotation. As the Java language runtime supports annotations our method can be exposed as a service interface by simply annotating the class.

```
@Service(MortgageCalculator.class)
public class MortageCalculatorImpl implements MortageCalculator {

    public double getMonthlyPayment(double principal, float interestRate) {
        ...
    }
}
```

The @Service annotation tells the SCA runtime that the MortgageCalculatorImpl class instances are exposed as services with an interface defined by the MortgageCalculator interface.

Component References

Now let's look at how a component references other components. We'll use the Loan Approval component that references other components as our example here. Loan Approval is implemented using the Java language and will use annotations. It uses @Reference to indicate its dependency on RiskAssessment, CreditCheck, InterestRateQuote, and MortgageCalculator. The referenced components can be local or remote and the SCA runtime will ensure that these references are correctly set at runtime based on the wiring found in the completed application's SCDL files (shown later in this article). See Listing 1.

Component Interfaces

The business functions provided by a service or required by a reference are described using interfaces in SCA. The interfaces represent the contract for a service or reference. Java and WSDL are two typical interface definition languages.

```
@Remotable
public interface CreditCheck {
    int getCreditScore(String ssn);
}
```

Raymond Feng is a senior software engineer with IBM. He is now working on the Service Component Architecture (SCA) runtime implementation in Apache Tuscany project as a committer. Raymond has been developing SOA for more than 4 years and he was a key developer and team lead for WebSphere Process Server products since 2002.

rfeng@us.ibm.com

OpenLaszlo

Advancing the Web Experience™

Hit the road to freedom with OpenLaszlo, the only open source advanced Ajax development platform that lets you choose between Flash® and DHTML for running your applications.

Open Source:
Create and control cost-effective, mission-critical web applications

Open Standards:
Build web applications with the most mature and advanced AJAX (JavaScript and XML) development platform

Open Architecture:
Designed to support multiple runtimes, including Flash, DHTML and embedded devices (mobile and TVs)

Laszlo®

Download today at www.openlaszlo.org

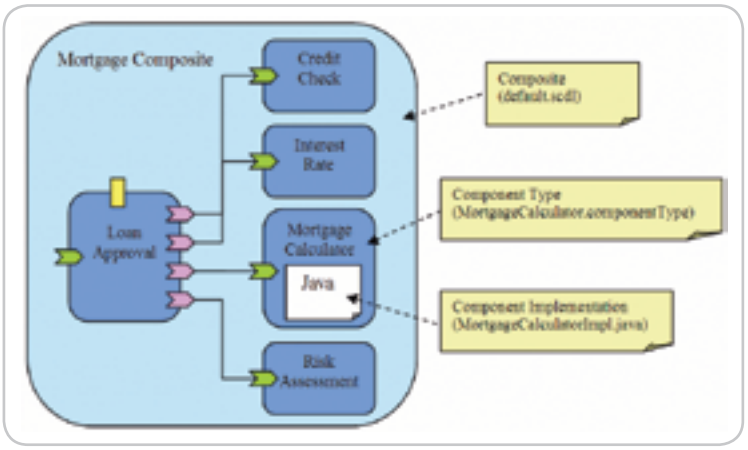


Figure 4 Describing components

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0" name="MortgageComposite">
    <component name="LoanApprovalComponent">
        <implementation.java class="mortgage.LoanApprovalImpl" />
        <property name="minimumCreditScore">600</property>
        <reference name="creditCheck">CreditCheckComponent</reference>
        <reference name="interestRateQuote">InterestRateQuoteComponent</reference>
        <reference name="riskAssessment">RiskAssessmentComponent</reference>
        <reference name="mortgageCalculator">MortgageCalculatorComponent</reference>
    </component>
    <component name="CreditCheckComponent">
        <implementation.java class="mortgage.CreditCheckImpl" />
    </component>
    <component name="InterestRateQuoteComponent">
        <implementation.java class="mortgage.InterestRateQuoteImpl" />
    </component>
    <component name="RiskAssessmentComponent">
        <implementation.java class="mortgage.RiskAssessmentImpl" />
    </component>
    <component name="MortgageCalculatorComponent">
        <implementation.java class="mortgage.MortgageCalculatorImpl" />
    </component>
</composite>
```

Figure 5 Wiring components together in the composite file

Interfaces can be local or remotable. Local interfaces are the most optimized for local interactions between components in the same composite. In contrast, remotable interfaces can be used for loosely coupled remote interactions.

Some business services have peer-to-peer relationships that require a two-way dependency at the service level. In these cases, the business service represents both a consumer of a service provided by a partner business service and a provider of a service to the partner business service. This is especially the case when the interactions are based on asynchronous messaging rather than on remote procedure calls. SCA uses bi-directional interfaces to directly model peer-to-peer bi-directional business service relationships.

For some services a sequence of operations must be called to achieve some higher-level goal. The sequence of operations is referred to as conversation. If the service uses a bi-directional interface, the conversation may include both operations and callbacks. SCA allows interfaces to be marked as conversational to bracket the series of operations in the same conversation.

Component Properties

Component properties can be used to alter the behavior of a component at runtime without making code changes. Let's assume that the LoanApproval component has a component property called "minimumCreditScore," which can be set to different values based on company policy. Below is a code snippet from the LoanApproval component implementation that uses an @Property annotation to identify a property called minimumCreditScore. The property has a default value of 650:

```
private int minimumCreditScore = 650;

// Property declaration using a setter method
@Property(name = "minimumCreditScore", override = "may")
public void setMinimumCreditScore(int minimumCreditScore) {
    this.minimumCreditScore = minimumCreditScore;
}
```

The following illustrates customization of the component by setting the "minimumCreditScore" property to 600 in the composite SCDL file to override the default value (650) defined in the component type (remember that we're using Java language annotations to define the component type):

```
<component name="LoanApprovalComponent">
    <implementation.java class="mortgage.LoanApprovalImpl" />
    <property name="minimumCreditScore">600</property>
    ...
</component>
```

Composites - Composing Components

So far we've concentrated on developing individual components, making them available as services and defining their dependencies on other services. Now let's look at how the components can be assembled to provide a business solution. This is referred to as a composite, which is a logical concept. A composite contains one or more components (see Figure 5).

If we look at the composite file (default.scdl) for MortgageComposite we can see how this draws all of the components together.

The SCA runtime uses the information in this SCDL file to instantiate, assemble, and configure the components. As can be seen from the example each component is identified by a <component> element in the file and can have references to other components. In this example, LoanApprovalComponent has four <reference> elements that are wired to four other components in the composite. The wiring is depicted through arrows in the diagram. The interfaces on both sides of the wire have to be compatible.

A composite can be reused as a component in the assembly but we don't show an example in this article.

Local Services

The composite file we've just seen shows how component references are "wired" to other components. There's no information included in this composite file to describe what techniques should be used to pass messages between the components. In this case SCA assumes that the components will be local to one another, i.e., they'll be instantiated and run in the same process address space. As we're using the Java language in this example the component instances will run in the same Java VM. SCA is free in this case to use the most efficient mechanism for moving a message from one component to another. This is likely to be a direct component-to-component call with little or no mediation.

On the face of it composition of components using local wiring may not appear to be very useful. Why not simply code these components as normal Java classes and have them interact in the normal way? In the case of coarse-grain components SCA has a number of advantages.

- Components can easily be reused and reconfigured in other compositions
- Components that are local today can be made remote tomorrow
- Components implemented using different supported programming languages can easily be assembled

Remote Services

A remote service could be running in a different process on the same physical computer or on a different computer. In the loan approval example we'll now show how it can interact with remote services. We'll first expose CreditCheck as a service in its own right and then we will replace it with an externally provided CreditCheck Web service.

Exposing Components as Remote Services

Let's expand our scenario. Suppose the MostMortgage Company has a business partner who is interested in using the CreditCheck functionality. In response to that, the CreditCheck component can be made into a Web Service that can be accessed by the LoanApproval component or by the business partner. With SCA, this is surprisingly easy and involves simply splitting off the CreditCheck component into a new composite, adding a service element to this new composite file and then wiring the service element to the CreditCheck component that provides the implementation of the service (see Figure 6).

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0"
            xmlns:wsdli="http://www.w3.org/2006/01/wsdl-instance"
    name="CreditComposite">

    <service name="CreditCheckWebService">
        <interface.wsdl interface="http://credit#wsdl.
          interface(CreditCheck)"
                  wsdl:wsdlLocation="http://credit wsdl/credit.wsdl" />
        <binding.ws endpoint="http://credit#wsdl.endpoint(CreditCheck
          Service/CreditCheckSoapPort)"
                  conformanceURIs="http://ws-i.org/profiles/basic/1.1"
                  location="wsdl/credit.wsdl" />
        <reference>CreditCheckServiceComponent</reference>
    </service>

    <component name="CreditCheckServiceComponent">
  <implementation.java class="credit.CreditCheckImpl"/>
    </component>

</composite>
```

Note that the new service element references a WSDL file to both describe the service interface and describe the binding.

Replacing Local Components with Remote Services

In the previous section we made CreditCheck a remote component. Now we show how MortgageComposite can be changed to reference this remote Web Service. All we have to do is change the SCDL file for the MortgageComposite. We add a reference element named CreditCheckReference to declare the remote CreditCheck Web Service and we rewire the creditCheck reference on the LoanApprovalComponent to this new reference.

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0" name="Mortgag
eComposite">
```

```
    <component name="LoanApprovalComponent">
        ...
        <reference name="creditCheck">CreditCheckReference</refer-
ence>
        ...
    </component>

    <reference name="CreditCheckReference">
        <interface.java interface="mortgage.CreditCheck" />
        <binding.ws endpoint="http://credit#wsdl.endpoint(CreditChe
ckService/CreditCheckSoapPort)"
            location="wsdl/credit.wsdl" />
    </reference>

    ...

</composite>
```

Note that the reference in SCDL specifies a binding called "binding.ws". The binding information indicates the protocol used to send messages across the wire. The system integrator here has the flexibility to configure any binding that's appropriate to enable secure communication between MortgageComposite and CreditComposite. Our system diagram is now updated as shown in Figure 7.

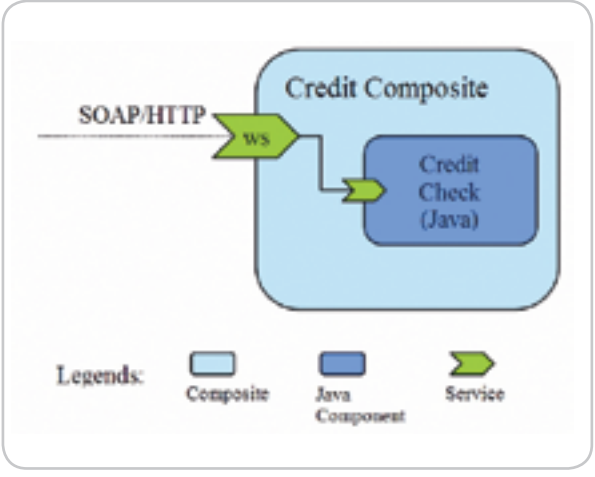


Figure 6 The check credit composite

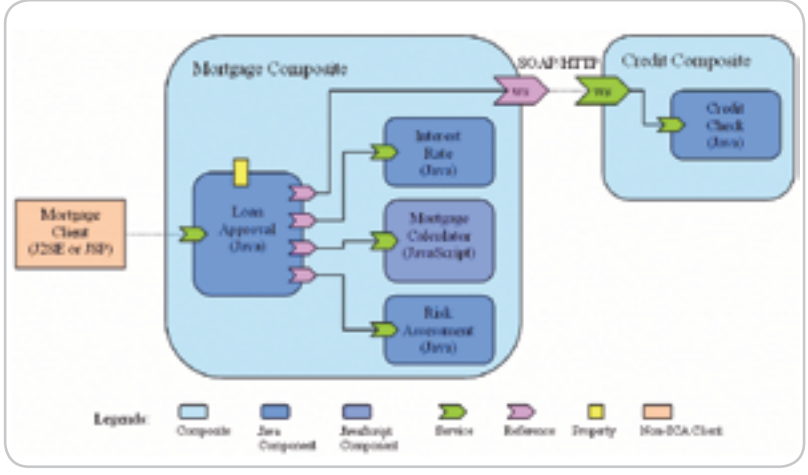


Figure 7 Connecting the MortgageComposite to the CheckCreditComposite

The CreditCheck Web Service isn't required to be implemented as an SCA component. Suppose that there's an existing CreditCheck Web Service offered by a third party. We can take the same approach to call this Web Service. The only difference is that the WSDL would contain the URL of the hosted Web Service. In the next diagram, the CreditCheck Web Service is provided as an Axis2 service and is deployed in an Apache Tomcat server (see Figure 8).

```
<composite xmlns="http://www.osoa.org/xmlns/sca/1.0" name="Mortgag
eComposite">

    <component name="LoanApprovalComponent">
        ...
        <reference name="creditCheck">CreditCheckReference1</refer-
ence>
        ...
    </component>
    <reference name="CreditCheckReference1">
        <interface.java interface="mortgage.CreditCheck" />
        <binding.ws endpoint="http://credit#wsdl.endpoint(CreditChe
ckService/CreditCheckSoapPort1)"
            location="wsdl/credit.wsdl" />
    </reference>
    ...
</composite>
```

Multi-Language Extensions

So far we've talked about Java language components and local and remote services but we've talked very little about the alternatives. You may have realized by now that all SCA does, through its SCDL, is describe components and the way that they can be assembled. It doesn't mandate how those components will be implemented or indeed how messages will be passed along the wires that join one component to another. Runtime implementations are free to add new implementation extensions. The Apache Tuscany project has provided extension APIs to make providing new implementation-type support as easy as possible. The Apache Tuscany Java technology SCA runtime implementation currently supports the following implementation types: implementation.java, implementation.javascript, and implementation.ruby. Using the extension mechanism the Tuscany community is currently working on Spring and BPEL implementation types.

The Apache Tuscany C++ SCA runtime implementation currently supports the following implementation types: implementation.cpp, implementation.python, and implementation.ruby.

Currently the PHP SCA runtime implementation only supports SCA components implemented in PHP.

You may have noticed in Figure 8 that we've changed the MortgageCalculator component from Java to use the JavaScript container from the Apache Tuscany Java SCA runtime.

ALL ATTENDEES RECEIVE: Certificate of Completion PLUS Course Materials on DVD!*

AJAX ONE-DAY HANDS-ON INTENSE TRAINING!

AJAXWorld University's **"AJAX Developer Bootcamp"** is an intensive, one-day hands-on training program that will teach Web developers and designers how to build high-quality AJAX applications from beginning to end. The AJAX Developer Bootcamp is intended to be the premier AJAX instructional program presently available anywhere.

AJAXWORLD UNIVERSITY BOOTCAMP

www.AJAXBOOTCAMP.sys-con.com

Roosevelt Hotel
New York, NY
January 22, 2007

Roosevelt Hotel
New York, NY
March 18, 2007

What You Will Learn...

Overview of AJAX Technologies
– HTML vs. DHTML
– Network Concerns
– Asynchronous Conversations with Web servers
– The characteristics of high-quality AJAX applications
The Web page is the application
What the server provides
User interaction

Understanding AJAX through the basics of AJAX
– Asynchronous server communication
– Dynamic HTML,
– Javascript Design patterns
– User interface strategies for building elegant, highly addictive Web sites and applications
– The Essential AJAX Pieces
Javascript
Cascading Style Sheet (CSS)
Document Object Model (DOM)
XMLHttpRequestObject
– The AJAX Application with Javascript
– Using CSS
– Structuring the View Using the DOM
Applying Styles with Javascript
Communicating with the Web Server in the Background
Designing AJAX Applications
Design Patterns
– Introduction to AJAX Frameworks-
Dojo, script.aculo.us, Prototype
Over of framework capabilities
– Examples of frameworks in use
– Best Practices

Hand-On Development The Fundamentals: Building the Framing for an Ajax Application
– Review the courseware code with the Instructor
– Begin building a working AJAX application and start applying technique and technologies as introduced in class
– Create the basic AJAX application by creating HTML, Javascript, and CSS files
– Learn Best Practices and Validation
– Learn and add script.aculo.us effects
– Learn and add the Dojo Framework

Adding Basic Ajax Capabilities to a Web Page: Going Deep Into the AJAX User Experience
– Elements on the Rich Internet Experience
Interactivity
Robustness
Simplicity
Recognizable Metaphors
Preservation of the Browser Model Bookmarks/Back Button
– Background operations
– Building a AJAX Notification Framework
– Provenance and Relevance
– Rich Experience Support with Third-Party AJAX Client – Framework
– Using AJAX layouts, containers, and widgets
– Patterns for Animation and Highlighting
– User Productivity Techniques
– Tracking Outstanding Network Requests

Hands-On Development: Expand the Application with more Advanced Ajax
– Review the courseware code with the Instructor
– Expand the Mural Application
– Add Features using Dojo
– Add specific Dojo Libraries to support Ajax widgets
– Work on server side communications in the background
– Create a tabbed layout
– Create a submission form to upload to the server, all without reloading the page
– Create an Ajax submission form that will take uploads on one tab.
– Created a form validation that ensures only the right information is submitted.
– More Stretch work for those who want to learn additional concepts

Advanced AJAX Concepts
– Review Ajax Concepts
– SOA and Mashups
– Current state of Ajax Frameworks
– Web 2.0 and the Global SOA
– Ajax Constraints
– Design Patterns
– Javascript Timers
– Ajax Programming Patterns
– Performance and Throttling

Hands-On Development: Working with Advanced Ajax Capabilities
– Review the courseware code with the Instructor
– Work with the Accordion control
– Learn how to use the Tree control
– Explore Dojo's animation capabilities
– Explore how the debug output can be used in elements
– Tour Dojo and RICO demos
– Experiment with new Dojo features from the Dojo demos source code and attempt to add them to various parts of the Mural application
– Overview of Future of AJAX and Rich Internet Applications

*ALL RELEVANT COURSE MATERIALS WILL BE OFFERED ON DVD AFTER THE EVENT

HURRY! REGISTER NOW FOR EARLY-BIRD DISCOUNT!

What Attendees Are Saying...

"The trainer was excellent. The material too!"

"The hands-on, although long, was useful and educational!"

"The instructor was good. He answered questions thoroughly!"

"Well designed and organized. Good mix of lecture vs lots of hands-on!"

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

The JavaScript implementation of MortgageCalculator.getMonthlyPayment() is as follows:

```
function getMonthlyPayment(principal, years, interestRate) {
    var monthlyRate = interestRate / 12.0 / 100.0;
    var p = Math.pow(1 + monthlyRate, years * 12);
    var q = p / (p - 1);
    var monthlyPayment = principal * monthlyRate * q;
    return monthlyPayment;
}
```

The component-type file (no changes required):

```
<componentType xmlns="http://www.osoa.org/xmlns/sca/1.0" xmlns:
xsd="http://www.w3.org/2001/XMLSchema">

    <service name="MortgageCalculatorService">
        <interface.java interface="mortgage.MortgageCalculator"/>
    </service>

</componentType>
```

We can now change the Mortgage composite to reference the Javascript MortgageCalculator component as follows:

```
<component name="MortgageCalculatorJSComponent"
           xmlns:js="http://incubator.apache.org/tuscany/xmlns/
container/js/1.0-incubator-M2">
    <js:implementation.js script="MortgageCalculator.js" />
</component>
```

Bindings and Extensibility

The protocols used to transfer messages from one component to another are also extensible. As an example, the Apache Tuscany runtime provides explicit extension APIs to aid the building of new bindings. The Apache Tuscany Java SCA runtime implementation currently supports the following bindings: binding.ws, binding.celtix, binding.jsonrpc, and binding.rmi.

As you have seen, SCA defines an assembly model that can leverage existing technologies. We used the Java and JavaScript languages and communicated with external Web Services. This means SCA can be introduced incrementally into an organization without having to rewrite or replace existing applications. The evolving list of implementation types and bindings available in Apache Tuscany is representative of commonly used technologies and is constantly growing. The implementation and binding extension APIs can be used to build specific extensions where support isn't provided by default.

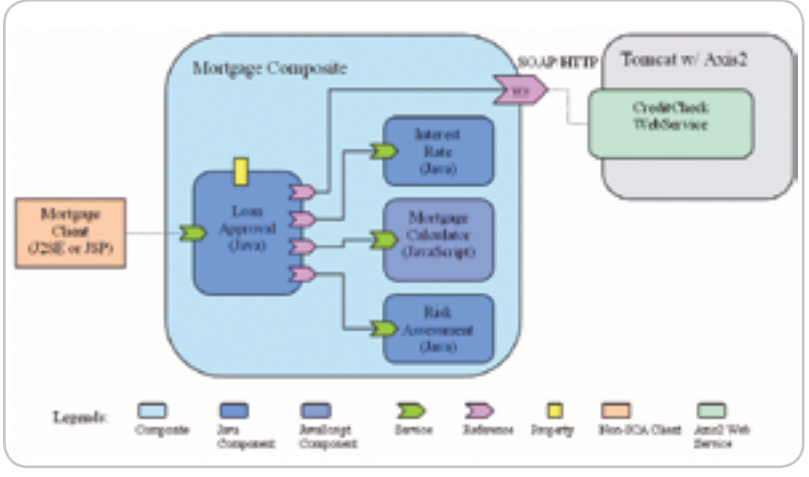


Figure 8 Connecting the MortgageComposite to a non-SCA Web Service

Quality of Service and Policy

SCA cleanly separates component implementation from the mechanisms that components use to exchange messages. This separation also allows the policies for message transmission to be stated independently of component implementation. So if you require messages transmitted, for example, reliably, within a transaction context or in encrypted form then the addition of statements of policy to the SCDL files will support this. The specifications for these policy statements aren't complete yet but you can see draft versions up on the OSOA site (http://www.osoa.org/display/Main/Service+Component+Architecture+Specifications).

Accessing SCA Services from Non-SCA Clients

In our MostMortgage example we've already shown that our SCA-based system can expose a service interface for others to call and can call service interfaces provided by others.

In the latter case SCA can call any service that's reachable using a protocol that the SCA runtime supports. We've shown how the CreditCheck Web Service can be provided by some third party and it need not be implemented using SCA. In our example we relied on accessing the WSDL description of the service and used a Web Service (SOAP/HTTP) binding to talk to it.

We haven't demonstrated yet how a non-SCA client can talk to SCA services. In our example we used SCA to expose the CreditCheckComponent for others to call via Web Services. Any Web Service-capable client could then call our service regardless of the technology used to implement the client. SCA does however provide APIs for non-SCA clients to invoke an SCA service locally.

```
public class MortgageClient {
    public static void main(String[] args) throws Exception {

        // Locate the service using SCA APIs
        CompositeContext context = CurrentCompositeContext.getCon-
text();
        LoanApproval loanApplication = context.locateService(Loan
Approval.class,

 "LoanApplicationComponent");
        ...
        // Invoke the service
        boolean result = loanApplication.approve(customer,
200000d, 30);
    }
}
```

> SCA provides a concise and flexible model for describing and developing SOA applications and addresses the strategic requirements demanded by agile IT environments"

Note that the SCA API class CurrentCompositeContext allows a service to be located by name. This is the component name as it appears in the composite file. The result of this location step is a service instance or proxy that implements the service interface. When a method is invoked by the client code (approve() in this case), the SCA runtime will dispatch the operation parameters to the correct method in the component implementation according to the definitions in the SCDL file.

SDO and DAS

This article has focused on how the Service Component Architecture allows components to be described and assembled into working compositions. There's also a sister technology, Service Data Objects (SDO), that describes a common API for accessing data. This is important to SCA because it allows the Apache Tuscany and PHP SCA runtimes to provide a common API for accessing the messages that arrive at, or are sent from, components. SCA and SDO work well together but can, of course, be used independently. More information and specifications for SDO can be found on the OSOA site (http://www.osoa.org/display/Main/Service+Data+Objects+Home).

The PHP SDO PECL extension provides an implementation of SDO for PHP. The Apache Tuscany project provides implementations of SDO for the Java language and C++. Both projects provide an implementation of a Data Access Service (DAS) that integrates the SDO data model with data storage systems such as relational databases.

Summary

SCA provides a concise and flexible model for describing and developing SOA applications and addresses the strategic requirements demanded by agile IT environments. The SCA programming model focuses on describing components and the way that they're assembled together. It's inclusive of existing technologies with a primary goal of operating well as an addition to existing heterogeneous environments.

This article aimed to provide a broad perspective of Service Component Architecture to intrigue the user to explore the technology further at (www.osoa.org) and experiment with the technology through three available free open source implementations, the Apache Tuscany (http://incubator.apache.org/tuscany/), SCA runtimes for the Java language and C++, and the SCA for PHP (http://www.osoa.org/display/PHP/SOA+PHP+Homepage) runtime.

Resources

- The Open Services Architecture specification site – http://www.osoa.org/display/Main/Home
- The Apache Tuscany incubator project – http://incubator.apache.org/tuscany/
- The PHP PECL SOA Project – http://pecl.php.net/package/sca_sdo and the homepage at http://www.osoa.org/display/PHP/SOA+PHP+Homepage
- The loan approval pplication examples – http://svn.apache.org/repos/asf/incubator/tuscany/sandbox/rfeng/samples.M2/mortgage/

Listing 1

```
@Service(LoanApproval.class) // Service declaration
public class LoanApprovalImpl implements LoanApproval {
    // Reference declarations using a protected or public field
    @Reference
    public  RiskAssessment riskAssessment;

    @Reference
    public  MortgageCalculator mortgageCalculator;

    @Reference
    protected InterestRateQuote interestRateQuote;

    // Reference declaration using a setter method
    private CreditCheck creditCheck;

    @Reference
    public void setCreditCheck(CreditCheck creditCheck) {
        this.creditCheck = creditCheck;
    }

    public boolean approve(Customer customer, double loanAmount,
int years) {
        int score = creditCheck.getCreditScore(customer.getSsn());
        if (score < minimumCreditScore) {
            return false;
        }
        float rate = interestRateQuote.getRate(customer.get-
State(), loanAmount, years);
        double monthlyPayment = mortgageCalculator.getMonthlyPayme
nt(loanAmount, years, rate);
        double ratio = monthlyPayment/customer.getMonthlyIncome();
        return riskAssessment.assess(score, ratio);
    }
}
```

Joe Winchester
Desktop Java Editor

Software Should Be More Hard Wearing

I am always in awe of people who develop hardware. They're the real engineers of our profession, the ones pushing forward the speeds at which things work, their size, and their connectivity. For example, in 2005 there were more computer chips produced worldwide than grains of rice harvested and at a lower unit cost. Tonight as I was watching a movie from the 1980s, instead of dating it by the big hair and shoulder pads, the tree rings were most visible by the size of the mobile phone the hero was using, the lack of a plasma or LCD widescreen TV in an otherwise luxurious living room, and the absence of a satellite navigation device as the lead characters got lost following directions from a map.

Why is it then that we in the software trade have let the side down so badly? Whereas hardware has advanced so dramatically in the last 40 years, I believe that software has stumbled along and, at worst, gone backward in a sort of fashion-driven and hysteria-led fervor. It must drive the hardware guys crazy – each time they achieve a new technological breakthrough with engineering brilliance, the software that runs on their new marvel seems to take the same number of steps backward that they managed to advance.

I remember coding in RPG in the 1980s and struggling to keep response times down so users wouldn't become dissatisfied with the application and bombard our help desk with complaints. If things got too bad, the customer could always upgrade their box to a newer and more powerful model and, riding the wave of Moore's Law with semiconductor speed doubling and price halving every six months, this was always the long stop for poorly performing code. However, as each new box came along with more memory and more processing power, the software somehow became larger and slower, so that the overall response time never seemed to really go down. I always felt sorry for the hardware engineers who with each new chipset probably thought, "Phew, we've just created an X mega/gigahertz chip; now everything that ran slowly before will perform OK, let's take a break," only to receive the call on their vacation that a new OS release or software package had been created that needed their new hardware specification as a minimum and could they please work on the next release to make it perform acceptably.

My mobile phone broke the other day after it fell on the kitchen floor, and after an apologetic call to my network provider, they agreed to send me a new one. Fortunately it wasn't going to cost me anything so I naturally talked myself into getting the latest whiz-bang model. When it arrived I was in awe of the thing: it's super slim, has the most gorgeous anodized case, a viewing area larger than my digital camera's, and as a piece of engineering is a work of art. The software on it, however, is absolutely terrible: to send a message I'm required to open a folder called messages, select "create," then type the message, pushing a button for "options", then pressing "send" again, which then asks me whether I want to send SMS or MMS. From a usability standpoint, it is just appalling. I don't care about network protocol; I want the phone's software to figure out which to use by analyzing the message's content.

There are two buttons on the front of the phone that will connect me to the Internet, one of which I keep pressing by accident when I'm backing up through menus. I don't want to connect to the Internet, ever, so I then have to press another button to cancel this. "Do you want to cancel?" This is the only time the phone ever asks for confirmation and, unfortunately for me, the only time I don't want to be asked. After pressing "Yes," it then comes up with a dialog telling me the number of bytes transferred that I have to press another button to dismiss. Why do I care about the number of bytes? It feels like the thing has a bunch of debug options left on. When I receive a call and the lid is open I press the green call button to accept. If I get a call and the lid is shut, all I have to do is open it and not push the button. Nice touch Mr. Usability Designer, except that if I open it and push the green button because I'm not thinking it holds the call. What is holding a call? All I know is that I can hear them and they can't hear me and I have to now push another button to retrieve the call.

My TV has a recordable receiver that can store shows on its internal hard drive allowing me to watch them at my leisure. This is great and totally changes the way one watches TV; however, when the chaps wrote their software on it they decided that the remote control buttons for changing channels would be up for a higher channel and down for a lower number. Not a bad choice, except the person writing the software to allow the program guide to be viewed used, not illogically, down for higher number channels (as the list is sorted with the lowest number at the top), so there are now two opposite ways to switch channel numbers depending on which part of the device's software you're using.

The list of frustrating software things that plague every existence doesn't end there, and it just disheartens me that the basic task of analyzing, understanding, and tooling for the user's most frequent and simple scenarios seem to have been overtaken by an obsession to cram as much functionality as possible into the hardware, overloading it with slow and poorly thought-through applications. In software we need to stop this millennium's ridiculous obsession with chasing the latest architectural fad or silver bullet that whizzes past, and instead focus on going back to basics and finding out what our users want and giving them something that's reliable, resilient, and more hard wearing.

Joe Winchester is a software developer working on WebSphere development tools for IBM in Hursley, UK.

joewinchester@sys-con.com

REGISTER TODAY AND SAVE!

Rich Internet Applications: AJAX, Flash, Web 2.0 and Beyond...

www.AjaxWorldExpo.com

AJAXWORLD™ EAST
CONFERENCE & EXPO

NEW YORK CITY

THE ROOSEVELT HOTEL LOCATED AT MADISON & 45th

SYS-CON Events is proud to announce the AjaxWorld East Conference 2007!

The world-beating Conference program will provide developers and IT managers alike with comprehensive information and insight into the biggest paradigm shift in website design, development, and deployment since the invention of the World Wide Web itself a decade ago.

The terms on everyone's lips this year include "AJAX," "Web 2.0" and "Rich Internet Applications." All of these themes play an integral role at AjaxWorld. So, anyone involved with business-critical web applications that recognize the importance of the user experience needs to attend this unique, timely conference – especially the web designers and developers building those experiences, and those who manage them.

BEING HELD MARCH 19 - 21, 2007!

We are interested in receiving original speaking proposals for this event from i-Technology professionals. Speakers will be chosen from the co-existing worlds of both commercial software and open source. Delegates will be interested in learning about a wide range of RIA topics that can help them achieve business value.

Enterprise Mashup Services

Part 1: Real-World SOA or Web 2.0 Novelties?

by Ric Smith

Since Web 2.0 kicked off scarcely a day goes by without a headline targeting mashups and their enablers, AJAX and Web Services, as the next hot Web technologies. Mashups are Web sites that integrate a variety of services (e.g., news feeds, weather reports, maps, and traffic conditions) in new and interesting ways. Just take a look at Zillow.com, which provides instant home valuations plotted as thumbtacks on a map (**Figure 1**), or HousingMaps.com, which marks listings from craigslist.org as captions on a map, and you'll get a clear picture of the power behind converging data sources.

Google Maps is often identified as the disruptive force that spawned the mashups movement. The popular mapping service is now the home of more than 600 mashups according to ProgrammableWeb.com. Why the hype? Google Maps provides a simple JavaScript API that makes geo-spatial data, a historically cost-prohibitive service, easily accessible to a broad audience with a variety of technical skills. Web 2.0-savvy developers are highly attracted to this simplified and accessible approach to SOA development because it no longer confines their Web site functionally to a user interface, but opens the site up to syndicate functionality and data. Thus a site's success is no longer based on traffic alone, but on the number of subscribers. The proliferation of these services is mind-boggling. ProgrammableWeb.com, a mashup-tracking site, has more than 300 registered services available to mix-and-match, and more than 1,100 registered mashups. Roughly 2.8 new sites are registered every day.

Despite the momentum behind mashups, corporate IT departments only consume a handful of the services used by the mashup crowd. Instead, businesses have constructed their own

ecosystem of services in parallel to the Web 2.0 movement. The problem with corporate adoption of mashups is twofold. First, the concept of mashups is often considered a social phenomenon or a grassroots effort tangential to enterprise software. Second, there's no clear path to integrate business services with those available on the Web. This second issue is due, in part, to the impedance mismatch between business services and mashup services. Mashup APIs are generally written in JavaScript, while services deployed to the enterprise are developed using Java or .NET technologies. Thus, both perception and technology define the barriers to corporations adopting an enterprise mashup strategy that incorporates external services such as Google Maps.

Despite these initial hurdles, the services and collaborative development style that mashups provide have created a buzz among enterprise software vendors. IBM recently announced a R&D effort to create an enterprise "mashup maker" — a tool that lets developers blend corporate services with external services, and rapidly assemble applications. More recently, Oracle announced the Oracle WebCenter Suite, which uses JSR-168/286 and WSRP 1.0/2.0 to mix-and-match corporate services using a combination of AJAX and Java portlets. The "mashup maker" and "mashups as a portal" are both interesting concepts and quite possibly the future of rapid business application development. In a later installment we will take a closer look at JSR-168/286 and WSRP 1.0/2.0 and how these standards can be used to create enterprise mashups that mix corporate services. Meanwhile, the aim of this article is to give application developers, specifically enterprise Java developers, the tools to build meaningful enterprise mashups that take advantage of the popular mashup services.

Ric Smith is a principal product manager for Oracle's Java/JEE/SOA tool offering, Oracle JDeveloper. Prior to joining the Oracle JDeveloper team, Ric worked for Oracle's consulting business as a principal consultant, where he specialized in Java EE architecture and development. Before his work in consulting, Ric was a lead software developer at Lockheed Martin. Ric holds a Bachelor's of Science in Computer Science with honors from the University of Arizona.

Figure 1 Zillow

Today's Toolkit

Business services developed with Java technologies generally fall into one of four categories: Enterprise JavaBeans, Spring, POJOs, or Web Services. The problem is that mashup APIs are generally implemented with JavaScript. There are a few services such as Flickr that offer parallel APIs written in Java, .NET, and a number of other technologies. For sites that don't offer a Java API, developers are left to their own devices to bridge the integration gap between JavaScript and Java. Existing Web Services provide the most direct path to integration, and processing XML over XMLHttpRequests, though tedious, is a relatively established method of binding AJAX-enabled interfaces to Web Services. There are a number of JavaScript packages, such as the vcXMLRPC Library, that help simplify the handling of XML-RPC requests. However, integrating Java-centric applications built with Spring or Enterprise JavaBeans makes for a more interesting problem. The first solution that comes to mind is to expose an existing EJB or Spring application as a Web Service, a simple task given that both EJB 3.0 and Spring 2.0 support remote method invocations via a Web Service end-point (JSR 181). Despite the simplicity of creating such a Web Service, we are still presented with the cumbersome task of processing XML with JavaScript.

Java-to-AJAX libraries such as Direct Web Remoting (DWR) and JSON-RPC-Java offer a simple alternative by marshaling Java objects to JavaScript and letting JavaScript communicate directly with server-side Java objects (Figure 2). So developers can interact with Java objects as if they were client-side JavaScript objects, negating the need to work with XML. For instance, by using a Java-to-AJAX library we can expose operations performed by a session bean to an AJAX-enabled Web interface and then combine the outcomes of those operations with a mashup service. An added benefit of the servlet architecture used by DWR and JSON-RPC-Java is that both libraries can take advantage of the authentication and session management provided by Java EE 5. Despite the similarities shared by both DWR and JSON-RPC-Java, DWR has a few advantages, two of which are the ability to handle recursive object structures and its integration with a large number

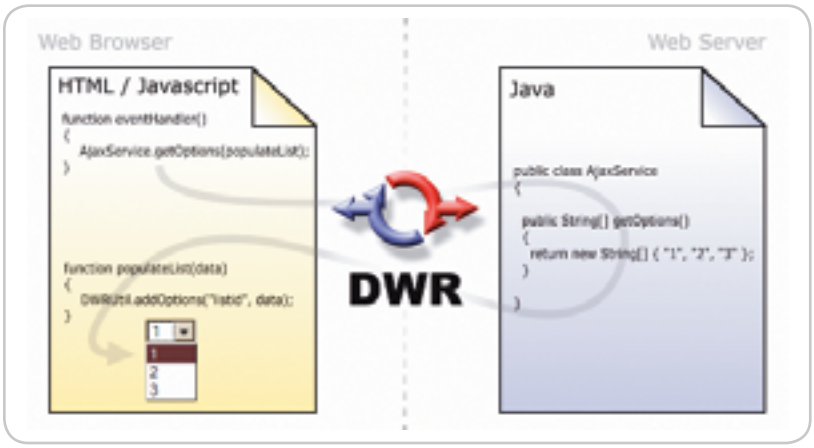


Figure 2 DWR JavaScript to Java interaction

of other libraries and frameworks such as Spring, Struts, JSF, Rife, WebWorks, and Hibernate. Thus, the amalgamation of DWR, enterprise Java services, and JavaScript mashup APIs blends the flexibility and creativity behind Web 2.0 with the reliability, scalability, and security of the Java EE architecture without needing to manipulate XML documents.

Building an Enterprise Mashup

The example application referenced in this article uses the Java Persistence API (JPA), Google Maps, and DWR to create a simple customer address book application (Figure 3). The application plots customers by address on a map using Google Map's JavaScript API. Users can update an entry by selecting a marker on a map and editing associated values in a form. Changes are committed by clicking the update, remove, and create. All records are stored in an Oracle database and persisted using the JPA.

The application provides a simple end-to-end example that demonstrates one method of blending external mashup services with Java-centric business services. Note that this article does not provide an in-depth look at DWR, JPA, or Google Maps. Instead, the intention is to prove the simplicity of integration between these technologies and enable you, the reader, to build your own enterprise mashups that leverage popular JavaScript APIs like Google Maps. Please refer to the references included in this article for more information on DWR, JPA, and Google Maps.

Setting up the DWR servlet is simple and can easily be added to an existing application. To install the servlet, first put

Figure 3 Example application

the DWR jar in the lib directory under the WEB-INF directory, and add the following lines to the web.xml descriptor:

```
<servlet>
 <display-name>DWR Servlet</display-name>
 <servlet-name>dwr-invoker</servlet-name>
 <servlet-class>
  uk.ltd.getahead.dwr.DWRServlet
 </servlet-class>
 <init-param>
  <param-name>debug</param-name>
  <param-value>true</param-value>
 </init-param>
</servlet>
<servlet-mapping>
 <servlet-name>dwr-invoker</servlet-name>
 <url-pattern>/dwr/*</url-pattern>
</servlet-mapping>
```

> Mashups are Web sites that integrate a variety of services (e.g., news feeds, weather reports, maps, and traffic conditions) in new and interesting ways"

Next, DWR must be made aware of the Java objects that should be remotable as JavaScript interfaces. The Customers object is the single JPA entity used in the application and maps to a Customers table (Listing 1). A session façade named JavaServiceFacade is used to encapsulate operations on the Customers entity object (Listing 2). Both the JavaServiceFacade and Customers classes must be registered with DWR to interact with each object via JavaScript. To do so, simply create a dwr.xml file in the WEB-INF directory as follows:

```
<dwr>
 <allow>
 <create creator="new"
          javascript="JavaServiceFacade">
 <param name="class"
  value="com…JavaServiceFacade"/>
  <include method="queryCustomersFindAll"/>
  <include method="queryCustomersFindById"/>
  <include method="mergeCustomers"/>
  <include method="persistCustomers"/>
  <include method="removeCustomers"/>
 </create>
 <convert converter="bean"

        match="com…Customers"/>
 </allow>
</dwr>
```

Figure 4 DWR test page

The dwr.xml descriptor defines a subset of methods on the JavaServiceFacade object to expose and declares a JavaBean converter to marshal the Customers entity as a return value or method parameter. In this example, two query methods as well as the merge, persist, and remove operations contained in the JavaServiceFacade are exposed. Once the dwr.xml file is created, three JavaScript files must be imported in the JSP or HTML pages contained in the application. This enables the use of the JavaScript representation or remote interfaces of the JavaServiceFacade and Customers Java objects. The imports are as follows:

```
<script type='text/javascript'
  src='../dwr/interface/
    JavaServiceFacade.js'></script>
<script type='text/javascript' src='../
  dwr/engine.js'></script>
<script type='text/javascript' src='../
  dwr/util.js'></script>
```

Figure 5 Methods accessible remotely

Note that only the JavaScript interface for the JavaServiceFacade is imported because it's the sole object that contains methods that are invoked explicitly. With the web.xml file configured, the dwr.xml file created, and all the necessary JavaScript files imported, the methods contained in the JavaServiceFacade are now accessible from a Web browser via JavaScript. To test the installation, open a Web browser and navigate to http://localhost:[port]/[nameofwebapp]/dwr. A screen should appear resembling that shown in Figure 4. Figure 5 shows all of the methods on the JavaServiceFacade that are accessible remotely.

The next step is to integrate the JavaScript interface for the JavaServiceFacade with the Google Maps API. To use the Google Maps API, an activation key is required. Visit http://www.google.com/apis/maps/signup.html to obtain a key. Keys are mapped to a unique URL so the key included in this article will only work with the source code provided. To use a key different from the one included in this example, simply add the following import statement to the top of your JSP page, replacing the key contained in the URL with the one obtained from the Google Maps registration page. The import below is required in the JSP page irrespective of the activation key used.

```
<script src="http://maps.google.com/maps?f
ile=api&amp;v=2&amp;key=value" type="text/
javascript"></script>
```

With both DWR and Google Maps configured, DWR calls to the JPA façade and the Google Maps API can be integrated. A simple load function and a div tag, in this case named "map,"

are used to initialize the mapping UI. The load function is assigned to the onload attribute of the body tag in the HTML or JSP page. Note that if you're using Dojo you must use the frameworks dojo.addOnLoad function instead of the onload attribute. The load function used in the example application not only initializes the Google Maps API, but also calls the queryCustomersFindAll function defined in the JavaServiceFacade JavaScript interface to retrieve all the Customers objects. The JavaServiceFacade.queryCustomersFindAll function accepts a reference to another function as an argument. The argument handles the return value of the JavaServiceFacade.queryCustomersFindAll function as an asynchronous callback. In this case the callback handler is the processCustomers function, which processes the list of Customers returned by the JavaServiceFacade.queryCustomersFindAll function, plotting each Customers object on the map as a marker (Figure 6).

```
// Called on intial page load
function load() {
 if(GBrowserIsCompatible()){
  // Remotely call Java method
  // to extract all Customers

  JavaServiceFacade.
   queryCustomersFindAll(processCustomers);
  // Initalize map object
  map =
   new GMap2(document.getElementById("map"));
  // Add navigation controls to map
  map.addControl(new GSmallMapControl());
  map.addControl(new GMapTypeControl());
  // Initalize geocoder
  geocoder = new GClientGeocoder();
 }
}

// Plots an array of Customers on
var processCustomers =
  function(customers){
 // test if array is null
 if(customers != null &&
  typeof customers == 'object'){
  // iterator over array of customers
  for(var i=0;i < customers.length; i++){
   // plot each customer on the map
   addMarkerForCustomer( customers[i] );
  }
 } else {
  alert("Customer record is null!");
 }
 // Set map center and magnification
 map.setCenter(
  new GLatLng(37.4419, -122.1419), 9);
}
```

Plotting markers with Google Maps requires three simple steps. First, a geospatial point on the map is obtained by using the Google geocoder API. Second, a GMarker object representing the geospatial is created. Finally, the marker is added to the map at the given point by calling the addOverlay function on the current instance of the GMap2 object – the object that represents the map displayed in the UI. Below is the function used to plot a Customers object as a marker. In the example a geo-spatial point is generated with the address information provided by the Customers object. A helper function is used to create a new GMarker, which is then added to the map object with a call to the addOverlay function.

```
// Plots a customer on the map as a marker
function addMarkerForCustomer(customer){
…
 // create address string
 var address = customer.address + …;
 // create point using geocoder
 geocoder.getLatLng(
  address,
  function(point) {
  …
  // create marker
  var marker = createMarker(point,
                            customer);
  // overlay marker on map
  map.addOverlay(marker);
  }
 );
…
}
```

The JavaScript functions used to update and insert entries make use of the addMarkerForCustomers function. The createCustomer, updateCustomer, and removeCustomer functions share similar code and for the sake of brevity only the create updateCustomer Customer function is described in this article. See Listing 1 for a detailed look at the createCustomer and removeCustomer functions.

The updateCustomer function calls a utility function, DWRUtil.getValues, provided by DWR that extracts values from HTML elements that contain ids that map to name/value pairs contained in a JavaScript object. In this example the DWRUtil.getValues function is used to populate a Customers JavaScript object that is persisted using the JavaServiceFacade.mergeCustomers function. The JPA merge operation updates the corresponding record in the database. The second argument passed to the JavaServiceFacade.mergeCustomers function is an anonymous function that's called after the Customers object is merged and is used to delete the currently selected marker. The deleted marker is replaced with a new one representing the changes to the Customers entity.

Figure 6 Customers plotted on a map

Figure 7 Selecting a marker

> With both DWR and Google Maps configured, DWR calls to the JPA façade and the Google Maps API can be integrated"

The removeSelectedMarker function is responsible for removing the currently selected marker on the map. The function uses the Google Maps API to close the caption for the selected marker and remove the marker from the overlay.

```
// Updates the currently selected customer
function updateCustomer(){
…
 DWRUtil.getValues(customer);
 // Remotely call Java method
 // to update record
 JavaServiceFacade.mergeCustomers(
  customer,
  function(){
  // remove selected marker
  removeSelectedMarker();
  // add new marker with updates
  addMarkerForCustomer(customer);
 });
…
}

// Removes the currently selected marker
function removeSelectedMarker(){
…
 // close the caption window
 map.closeInfoWindow();
 // remove the current marker
 map.removeOverlay(currentMarker);
…
}
```

The updateCustomer function and similarly the createCustomer and removeCustomer functions are invoked when a user clicks a form button on a page. Attaching the createCustomer function to the onclick event fired by the button when it's clicked makes the association between the button and the updateCustomer function. Thus, a single click of an HTML form button creates a Customers JavaScript object that's converted to a JPA entity and persisted to a database.

```
<button id="update"
        onclick="updateCustomer();">
  Update
</button>
```

Clicking a marker on the map populates the customer form in the application (Figure 7). This use case exercises the GEvent facility in the Google Maps API, which is used to assign action listeners to objects plotted on the map. A listener that responds to a mouse click on a marker is created in the createMarker function. The assignment is made using the following code. Note that the function created to handle the event populates the customer form using the DWR utility function DWRUtil.setValues, which assigns values to elements contained in the page that map to the name/values pairs defined in a JavaScript object. The event handler also opens an information window on the map that displays the selected customer's name and address (Figure 8).

```
GEvent.addListener(marker, "click",
  function() {
 // set the selected marker
 currentMarker = marker;
 // set form field values
 DWRUtil.setValues(customer);
 …
  // open info window with customer
  // name and address
  marker.openInfoWindowHtml(…);
});
```

In summary, there are six essential steps to set up both DWR and Google Maps or any other JavaScript mashup API in a Web application:

1. Edit the web.xml file and copy the dwr.jar file to the WEB-INF/lib directory.
2. Create the Java objects that will be exposed remotely as JavaScript interfaces.
3. Enable DWR to expose objects and methods remotely by defining a dwr.xml file.
4. Acquire an activation key.
5. Import all JavaScript files into your JSP pages.
6. Finally, write a little JavaScript. Go on it's not so bad!

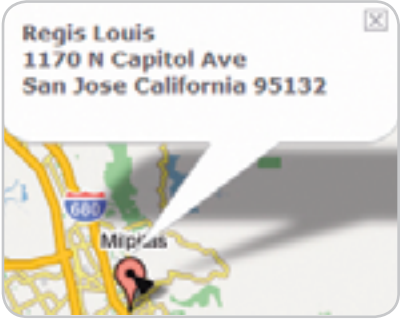


Figure 8 Caption containing customer name and address

Conclusion

In this article you learned how to integrate Java services with mashup APIs implemented in JavaScript. The DWR framework enabled this simple integration by providing remote access to server-side Java objects via JavaScript. The value of this example is that it illustrates the simplicity of incorporating services implemented in either Java or JavaScript in the context of an enterprise-ready application. Thus, we now have one of the tools necessary to build an enterprise mashup. Next time, we'll look at how to consolidate the JavaScript code that comprises your mashup application into components that can be reused in a variety of development environments. For more information on the technologies referenced in this article please refer to the references provided.

References

- Direct Web Remoting (DWR) framework: http://getahead.id.uk
- Google Maps API: http://www.google.com/apis/maps/documentation/
- Java Persistence API: http://java.sun.com/products/ejb/
- Information on Web 2.0, SOA, and Mashups: http://blogs.zdnet.com/
- JDJ Volume 11 Issue 10 article: "AJAX: The Easy Way" provides a detailed look at the DWR framework: http://java.sys-con.com/read/286892.htm

The World's Leading Java Resource Is Just a >Click< Away!

JDJ is the world's premier independent, vendor-neutral print resource for the ever-expanding international community of Internet technology professionals who use Java.

SYS-CON MEDIA

ONLY $69.99

ONE YEAR
12 ISSUES

Subscription Price Includes FREE JDJ Digital Edition!

www.JDJ.SYS-CON.com

or **1-888-303-5282**

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

Mediation and the Man in the Middle

Patterns for designing scalable and robust user-interfaces

by Michael Birken

Even for many seasoned developers, Swing code can be notoriously difficult to organize. Where is the right place to put parsing and validation logic? How do you prevent those threading issues that cause lockups or repainting glitches? Is it possible to unit test GUI logic? Can the code somehow be shared with other user-interfaces, like a web front-end? If these questions sound familiar, the solutions presented here may revolutionize the way you code with Swing.

Two-Layer Separation

Suppose you want to offer your end-users a Swing-based product built on top of a homegrown API, such as a mathematics package, that they can actually license for their own development purposes or perhaps even use to extend your product through a plugin mechanism. To achieve that goal, the API must be fully decoupled from all Swing code. Let's take a look at a simple example of such an API.

In the class Divider, I defined the following method:

```
public DivisionResult divide(
      int dividend, int divisor, IDivisionListener divisionListener)
      throws ArithmeticException { ... }
```

Given a dividend and a divisor it returns a DivisionResult, a simple bean containing quotient and remainder. If divisor is 0, it throws an ArithmeticException. I'll discuss the divisionListener parameter below.

Now, let's slap on a JFrame to exploit this API. DivisionFrame (see Figure 1) contains 2 JTextFields that enable the user to enter a dividend and a divisor. When the user presses the Divide button, the resultant quotient and remainder are displayed in a JLabel.

The simplest attempt at a separation of concerns is a GUI layer built directly on top of the API layer. The GUI layer consists of JFrames, JDialogs, Swing components and their data models. The API layer contains the business logic and as mentioned above, it should be possible to use the API layer without a GUI.

Since there are only 2 layers in this approach, user inputs must be prepared for the API layer in the GUI layer. In this case, Divider.divide() accepts integers, not strings. The ActionListener bound to the Divide button parses the values provided by the JTextFields and it either calls divide() or it changes the text color red to indicate invalid input.

Keep in mind that Divider actually represents an advanced mathematics package. Its methods may take several seconds or minutes to complete. I decided to simulate that effect by sleeping for 5 seconds inside of divide(). If divide() is called directly by the ActionListener, the GUI will appear frozen and possibly grayed-out for that time because of Swing's one-threaded nature (see the sidebar, The Event Queue).

The general solution is a request-response model. The GUI layer makes a request into the API layer on a new thread. That thread takes as long as is needed to complete the operation, freeing the event-dispatching thread to continue servicing the event queue. When the result of the operation is ready, the thread uses one of the static invokeXX() methods of the EventQueue class to safely update the GUI by requesting that the event-dispatching thread handle the update on its behalf.

In this model, a thread boundary exists between the layers: the event-dispatch thread is the only thread that runs within the GUI layer, worker threads run within the API layer, and neither type is allowed to cross the boundary. Unfortunately, there is no mechanism to automatically prevent a rogue thread from slipping through. Also, the code required to keep switching threads tends to be voluminous and ugly even with the aid of SwingWorker (see Resources).

Three-Layer Separation

Consider introducing a layer between the GUI and API layers to mediate data between them. This mediation layer is responsible for converting user input into a format acceptable by the API layer and for converting resultant values and other data from the API layer back into a format acceptable by the GUI layer. Ideally, the boundaries between the 3 layers should be formalized with interfaces; though, that may not be entirely possible if the API was provided by a third-party vendor. Finally, the event-dispatching thread is not permitted to cross into the mediation layer. Worker threads can pass back-and-forth between the mediation and API layers; however, they cannot enter the GUI layer.

Michael Birken is actively involved in the design and research of emerging trading technologies at a Manhattan-based financial software company. He's a Sun Certified Java programmer and developer. Michael holds a BS in computer engineering from Columbia University.

o__1@hotmail.com

For our simple example, I created a single class, aptly named Mediator, to serve as the entire mediation layer. To formalize the layers, Mediator implements IDivisionFrameMediator, which allows DivisionFrame to pass the values in the JTextFields directly as strings:

```
public interface IDivisionFrameMediator {
  public void divide(String dividendStr, String divisorStr);
}
```

For the reverse direction, DivisionFrame implements IDivisionFrame. It enables Mediator to push a DivisionResult back to DivisionFrame in the form of a string and to mark the input fields as valid or invalid:

```
public interface IDivisionFrame {
  public void showDivisionResult(String divisionResult);
  public void showValid(boolean dividend, boolean divisor);
}
```

Mediator is loosely-coupled to DivisionFrame; they each hold a handle to each other as an interface type. However, the handle cannot be a direct reference to the object in the opposing layer because that would break the thread-layer separation rules discussed above.

We could solve the problem by creating 2 proxy classes (see the sidebar, Proxies). Let's call these hypothetical classes MediatorProxy and DivisionFrameProxy. MediatorProxy implements IDivisionFrameMediator and contains a reference to Mediator. Whenever you invoke a method of MediatorProxy, it spawns off a new thread and uses it to call the corresponding method in Mediator. Similarly, DivisionFrameProxy implements IDivisionFrame and contains a reference to DivisionFrame. Calling a method of DivisionFrameProxy delegates the invocation to DivisionFrame using EventQueue.invokeLater().

With such proxies, the code in DivisionFrame and Mediator appears immaculate. DivisionFrame calls directly into the methods of its IDivisionFrameMediator and Mediator does the same with its IDivisionFrame. It's still a request-response model, but the proxies hide the thread switching details and there's no chance of a thread inadvertently slipping by. However, creating the proxies themselves is a tedious and repetitive task you shouldn't have to do yourself.

SwingProxy

To obviate the need to code the proxies by hand, I created a utility class called SwingProxy that dynamically generates them for you. SwingProxy provides a static method, newSwingProxy(), that accepts the target object and returns a new proxy that can be cast to any of the interface types that the target implements. For example, for an instance of DivisionFrame, which implements IDivisionFrame, you can create a proxy for it as follows:

```
IDivisionFrame divisionFrameProxy =
    (IDivisionFrame)SwingProxy.newSwingProxy(divisionFrame);
```

The proxy automatically takes care of the thread switching. In this case, if a method of divisionFrameProxy is invoked by a worker thread, it will call the corresponding method of divisionFrame with the event-dispatching thread. Similarly, a call by the event-dispatching thread will turn into a call by a worker thread.

The Event Queue

Swing provides a single thread, the *event-dispatching thread*, for servicing all user-interface requests, including repainting components. When you press a button, its associated ActionListener is not executed immediately. Instead, the request to execute it is placed onto the *event queue*. The event-dispatching thread pulls requests off the queue and executes them one-by-one, eventually handling the button press. If you implement a component listener that performs a time-consuming task or that makes a blocking call, the event-dispatching thread will sit there completely dedicated to it, neglecting the other requests on the event queue. The result is a non-responsive GUI that typically appears grayed-out because invalidated components make requests for repaint that also must pass through the event queue.

The event-dispatching thread is considered the only the thread that can safely access Swing components and their data models. A worker thread can request that the event-dispatching thread execute code on its behalf using invokeLater() or invokeAndWait(), static methods of the EventQueue class. Both methods accept a Runnable implementation where run() contains the code to execute. invokeLater() inserts the Runnable into the event queue and returns immediately. On the other hand, invokeAndWait(), enqueues the Runnable and waits until it is serviced by the event-dispatching thread before returning.

SwingProxy (see Listing 1) is built around java.lang.reflect.Proxy, a class that is capable of producing a proxy of a specified type at runtime. The Proxy utility produces proxies that resemble funnels; to create the proxy, you supply an implementation of the InvocationHandler interface and when you invoke any method of the proxy, it automatically gets funneled down to InvocationHandler's single method, invoke():

```
public interface InvocationHandler {
  public Object invoke(Object proxy, Method method,
      Object[] args) throws Throwable;
}
```

The first parameter is a reference to the proxy. The second is the method that was called in the form of a reflected method type. The third is an object array containing the arguments passed to the method.

SwingProxy contains 2 private inner classes. The first, CallHandler, implements InvocationHandler. Its invoke() method inspects the calling thread and switches accordingly. When it needs to delegate a call from a worker thread to the event-dispatching thread, it examines the return type of the method. If it returns void, invokeLater() is used. Otherwise, invokeAndWait() is called and the return value is passed back to the worker thread that called the proxy. When the proxy is called by the event-dispatching thread, the call is delegated to a worker thread and nothing is ever returned. The interface methods specifying calls from the GUI layer to the mediation layer should always return void. The worker threads that CallHandler use originate from a thread-pool supplied by java.util.concurrent.Executors.

Figure 1 DivisionFrame allows the user to enter a dividend and a divisor and get back a quotient and a remainder

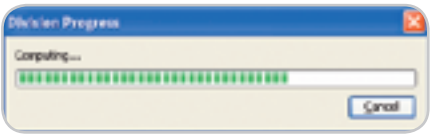


Figure 2 ProgressDialog displays the percent completed and allows the user to cancel the computation

Proxies

A proxy is an object that serves as a middleman for communication between a *source object* and a *target object*. Typically, the proxy and the target objects share the same interface. The source object holds a reference to target as the interface type, which makes the source oblivious to whether it is communicating directly with the target or communicating through the proxy.

Proxies are used throughout the Java APIs. Remote Method Invocation (RMI), for instance, allows a program to communicate with a target object on a remote machine and treat it as if it were a local object. The program actually holds a reference to a proxy that hides the networking details. The target object and the proxy share a common interface and the program only refers to the target as that type. The Java API for XML Web Services (JAX-WS), which supplants the Java API for XML-based Remote Procedure Call (JAX-RPC), performs a similar service over a different protocol.

The second inner class, TargetInvoker, implements Runnable, which allows it to be executed on a different thread. It performs the actual method invocation using reflection in its run() method.

Pushing Data to the GUI

The three-layer design described above is not limited a request-response model. The API can stream data to the GUI by pushing it through the mediation layer. To demonstrate this, I created a dialog to display the (artificially prolonged) progress of the division computation (see Figure 2).

ProgressDialog contains a JProgressBar to show completion percentage, a JLabel for status messages, and a button that allows the user to abort the computation. ProgressDialog lives within the GUI layer and to formally separate it from the other layers, it implements IProgressDialog:

```
public interface IProgressDialog {
  public void start();
  public void setProgress(String message, int progress);
  public void end();
}
```

Mediator holds a proxy to ProgressDialog as that type. start() makes the dialog appear, setProgress() updates the status label and the progress bar, and end() hides the dialog. In turn, ProgressDialog holds a proxy to Mediator in the form of an IProgressDialogMediator, which contains a method that is called when the Cancel button is pressed:

```
public interface IProgressDialogMediator {
  public void requestCancel();
}
```

Divider.divide() was written to abort if the executing thread is interrupted. Mediator obtains a reference to that thread before invoking Divider and requestCancel() simply interrupts it.

The third parameter of Divider.divide() is an IDivisionListener, which is repeatedly called back during the 5 second pause to simulate progress notifications:

```
public interface IDivisionListener {
  public void computationPerformed(int percentage);
}
```

Mediator implements IDivisionListener and it delegates the call to IProgressDialog.setProgress(). For time consuming methods that don't provide such a listener, JProgressBar can be put into indeterminate mode. That mode shows an animation conceptually similar to the moving logo in the corner of a web browser.

Mediator as the Controller

It is important to recognize that the mediation layer consists of more than just bidirectional adapters that convert data between formats. It contains the control logic that governs when and how the parts from the other layers are used. The control logic should not be intermingled with the adaptation logic that performs the parsing and the static validation.

In this simple example, Mediator.divide() contains most of the control logic (refer to Listing 2). As talked about above, divide() is invoked by the Divide button and it passes the user input fields as strings. Instead of attempting to parse the strings directly within divide(), they are used to create instances of a class called DivisionFrameParser. The constructor of DivisionFrameParser accepts a string field and parses it. The class provides methods to check if the parsing was successful and to retrieve the field as an integer. In this way, Mediator is focused on the interaction of classes across the layers and less on processing the shuttled data.

Alternate UIs

Mediator, DivisionFrame and ProgressDialog expose setters to enable their dependencies to be injected prior to use. The Main class, which serves as the entry point of the application, wires everything up. However, with Mediator loosely-coupled to the GUI layer, it's possible to completely replace the user-interface. To prove it, if you launch the application with a "-t" command-line argument, it will run in text-mode. Text-mode prompts the user for a dividend and divisor and it prints results back to the console. It was made possible by providing Mediator with alternate implementations of IDivisionFrame and IProgressDialog.

What about a web front-end? Web applications serve data on demand; data is not easily pushed to the browser. This makes showing progress updates, for instance, fairly tricky. As with text-mode, implementing a web front-end entails creating a new GUI layer, but it will also require additions to the mediation layer. The control logic discussed above was designed with pushing data in mind. New controller classes will be required for the request-response nature of the web; however, since we separated the parsing and validation logic from the control logic, it can be reused in a web application.

Testing

The GUI classes do not need to be connected to Mediator to launch them. To demonstrate this, I inserted a main() method into DivisionFrame that uses a mock implementation of IDivisionFrameMediator. In this case, the division request is simply printed out to the console.

Using mock implementations to represent the rest of the system is an especially useful technique if you are developing

without the aid of a GUI builder because it will enable you to quickly view your efforts without launching the complete application. Mock implementations can allow you to fully exercise the features provided by the GUI with much less effort. For example, you do not need to induce a problem in the real application just to make sure that error messages get displayed correctly.

Mock implementations are also the hallmark of automated unit testing. The online code that supplements this article includes JUnit tests for Divider, Mediator and DivisionFrameParser. To make Mediator easier to test, I made Divider implement IDivider, an interface that contains its single method. The complete execution cycle for Mediator is tested with mock implementations of IDivider, IDivisionFrame and IProgressDialog.

Real-Time Interaction

Suppose you want to alter DivisionFrame such that as you key in dividend and divisor, the text immediately turns red if it doesn't represent valid numerical data as opposed to after pressing the Divide button. You can bind a KeyListener to the JTextFields and receive an event for every keystroke, but where should you do the validation? One option is to delegate the key events to Mediator and let it validate and callback DivisionFrame; however, that cycle of execution is significantly different from the ones discussed above because the API layer is never invoked.

The mediation layer provides adaptation for the API layer and it shouldn't be used where the API is not needed. In this case, the behavior is entirely specific to the user-interface and nothing is gained by thread switching. DivisionFrameParser, which encapsulates the static validation logic, should be used directly inside of a KeyListener. The Mediator class remains the same and it double checks that the fields are valid also via DivisionFrameParser.

Sharing Objects

Consider a Swing application that contains a JTable with thousands of rows. Each row is a view into a simple bean where the columns are mapped to the bean properties. The table effectively enables the user to view and edit beans. The API layer requires access to a collection of those beans to function. It occasionally modifies bean properties and those changes should be reflected on the front-end.

In this example, which layer owns the beans? The beans are objects that can be manipulated by both the event-dispatching thread and worker threads apparently violating the layer-separation rules. Breaching the layers can be dangerous. For instance, if a worker thread were to remove a bean from the TableModel in the middle of a repaint, the event-dispatching thread could loop past the end of the collection and throw an exception.

Since a bean is simply a container whose methods execute in a timely manner, they can be shared by all layers with a little help. First, all methods of the bean should be synchronized to enable threads to access them atomically. For example, a worker thread should be able to obtain a lock on a bean and read several properties needed for a computation, knowing that all reads represent a consistent view. Though, it must be recognized that until that lock is released, the event-dispatching thread is potentially held up.

Second, when the API modifies a bean property, the JTable needs to be notified so it can repaint the associated cell. The bean requires a listener, such as java.beans.PropertyChangeListener, which is invoked by the setters. A mediator class would implement the listener and forward the event to the TableModel via SwingProxy to allow it to resolve the coordinates of the cell that requires repainting. When a cell is edited by the user, TableModel.setValueAt() is called. In that method, the event-dispatching thread would grab a lock on the bean, remove the listener to prevent the TableModel from being called back, update the property and restore the listener before releasing the lock.

Finally, the beans should originate from a factory class that provides a create() method and a destroy() method, and it would notify a mediator when either of those methods are called. In turn, the mediator would update the interested API collections and the TableModel in a thread-safe manner.

Listing 1

```
package example;

import java.lang.reflect.*;
import java.util.concurrent.*;
import java.awt.*;

public final class SwingProxy {

  private static class TargetInvoker implements Runnable {

    private Object target;
    private Object returnValue;
    private Throwable exception;
    private Method method;
    private Object[] arguments;

    public TargetInvoker(Object target, Method method,
       Object[] arguments) {
      this.target = target;
      this.method = method;
      this.arguments = arguments;
    }

    public boolean threwException() {
      return exception != null;
    }

    public Throwable getException() {
      return exception;
    }

    public Object getReturnValue() {
      return returnValue;
    }

    public void run() {
      try {
        returnValue = method.invoke(target, arguments);
      } catch(Throwable t) {
        exception = t;
      }
    }
  }

  private static class CallHandler implements InvocationHandler {

    private static final ExecutorService threadPool
        = Executors.newCachedThreadPool();
    private Class targetClass;
    private Object target;

    public CallHandler(Object target) {
      this.target = target;
      this.targetClass = target.getClass();
    }

    public Object invoke(Object proxy, Method method, Object[] args)
        throws Throwable {

      Method targetMethod = targetClass.getMethod(
            method.getName(), method.getParameterTypes());
```

We can get away with using DivisionFrameParser directly inside of a KeyListener because the parsing and validation occur almost instantly. If we required something more advanced, such as real-time spelling and grammar checking, then a call into the mediation layer is justified because that kind of validation will require a specialized API and it's unlikely to execute as timely. However, we must consider that every key press gets delegated to an independent thread. If you were to type too quickly, several threads will needlessly be processing the same input in parallel. To resolve this issue, we need to setup the KeyListener to take the request-response nature of the mediation layer into account. The KeyListener shouldn't call into mediation layer if it's already validating an input field in response to a prior key press; rather, the KeyListener should simply mark the field as changed. When the validation is complete and the GUI layer is called back, that invocation can check if the field has changed and make a successive call into the mediation layeraccordingly.

Conclusion

I hope you find the patterns discussed here useful in your own development efforts. The full source code including unit tests for the division application can be found at the online version of this article at http://jdj.sys-con.com. DivisionFrame and ProgressDialog were created using NetBeans 5.0. Refer to the NetBeans documentation for using the Swing Layout Extension library outside of NetBeans.

Resources

- Mediator pattern: http://en.wikipedia.org/wiki/Mediator_pattern
- Event-dispatching: http://java.sun.com/docs/books/tutorial/uiswing/misc/threads.html
- SwingWorker: https://swingworker.dev.java.net/
- Dynamic proxies: http://java.sun.com/j2se/1.3/docs/guide/reflection/proxy.html
- Reflection: http://java.sun.com/docs/books/tutorial/reflect/index.html
- JUnit: http://www.junit.org/index.htm
- NetBeans: http://www.netbeans.org/

```
      TargetInvoker targetInvoker = new TargetInvoker(
          target, targetMethod, args);
      if (EventQueue.isDispatchThread()) {
        threadPool.execute(targetInvoker);
      } else if (method.getReturnType() == void.class) {
        EventQueue.invokeLater(targetInvoker);
      } else {
        EventQueue.invokeAndWait(targetInvoker);
        if (targetInvoker.threwException()) {
          throw targetInvoker.getException();
        } else {
          return targetInvoker.getReturnValue();
        }
      }

      return null;
    }
  }

  private SwingProxy() {
  }

  public static Object newSwingProxy(Object target) {
    return Proxy.newProxyInstance(
        SwingProxy.class.getClassLoader(),
        target.getClass().getInterfaces(),
        new CallHandler(target));
  }
}
```

Listing 2

```
package example;

public class Mediator implements IProgressDialogMediator,
    IDivisionFrameMediator, IDivisionListener {

  private IDivider divider;
  private IDivisionFrame divisionFrame;
  private IProgressDialog progressDialog;
  private Thread divideThread;

  public void setDivider(IDivider divider) {
    this.divider = divider;
  }

  public void setDivisionFrame(IDivisionFrame divisionFrame) {
    this.divisionFrame = divisionFrame;
  }

  public void setProgressDialog(IProgressDialog progressMonitor) {
    this.progressDialog = progressMonitor;
  }

  public void divide(String dividendStr, String divisorStr) {
    DivisionFrameParser dividendParser
        = new DivisionFrameParser(dividendStr);
    DivisionFrameParser divisorParser
        = new DivisionFrameParser(divisorStr);
    divisionFrame.showDivisionResult("");
    divisionFrame.showValid(dividendParser.isFieldValid(),
        divisorParser.isFieldValid());

    if (dividendParser.isFieldValid()
        && divisorParser.isFieldValid()) {
      try {
        progressDialog.start();
        synchronized(this) {
          divideThread = Thread.currentThread();
        }
        DivisionResult divisionResult = divider.divide(
            dividendParser.getField(),
            divisorParser.getField(), this);
        synchronized(this) {
          divideThread = null;
        }
        if (divisionResult == null) {
          divisionFrame.showDivisionResult("[Cancelled]");
        } else {
          divisionFrame.showDivisionResult(
            divisionResult.getQuotient()
              + " R " + divisionResult.getRemainder());
        }
      } catch(ArithmeticException e) {
        divisionFrame.showDivisionResult("NaN");
      } finally {
        progressDialog.end();
      }
    }
  }

  public synchronized void requestCancel() {
    if (divideThread != null) {
      divideThread.interrupt();
    }
  }

  public void computationPerformed(int percentage) {
    progressDialog.setProgress("Computing...", percentage);
  }
}
```

JAVA DEVELOPER'S JOURNAL Advertiser Index

Advertiser	URL	Phone	Page
AjaxWorld East Conference 2007	www.ajaxworldexpo.com	201-802-3022	47
AjaxWorld University Bootcamp	www.ajaxworldbootcamp.sys-con.com	201-802-3022	43
Altova	www.altova.com	978-816-1600	Cover II
Backbase	www.backbase.com/jsf	866-800-8996	19
Business Objects	www.businessobjects.com/devxi/misunderstood		11
IBM	ibm.com/takebackcontrol/flexible		7
ICEsoft Technologies	www.icesoft.com	877-263-3822	33
Infragistics	www.infragistics.com/jsf	800-231-8588	8-9
InterSystems	www.intersystems.com/jalapeno3p		4
IT Solutions Guide	www.itsolutions.sys-con.com	888-303-5282	59
Java Developer's Journal	www.jdj.sys-con.com	888-303-5282	53
Jinfonet Software	www.jinfonet.com/live	240-477-1000	25
Laszlo	www.openlaszlo.org		39
Northwoods Software Corp.	www.nwoods.com	800-434-9820	41
OPNET Technologies, Inc.	www.opnet.com/panorama	240-497-3000	15
Parasoft Corporation	www.parasoft.com/jdjmagazine	888-305-0041	Cover IV
Recursion Software	www.recursionsw.com	800-727-8674	29
Software FX	www.softwarefx.com	800-392-4278	Cover III
SYS-CON Website	www.sys-con.com	888-303-5282	61
TIBCO Software Inc.	http://developer.tibco.com/	800-420-8450	17

General Conditions: The Publisher reserves the right to refuse any advertising not meeting the standards that are set to protect the high editorial quality of ***Java Developer's Journal***. All advertising is subject to approval by the Publisher. The Publisher assumes no liability for any costs or damages incurred if for any reason the Publisher fails to publish an advertisement. In no event shall the Publisher be liable for any costs or damages in excess of the cost of the advertisement as a result of a mistake in the advertisement or for any other reason. The Advertiser is fully responsible for all financial liability and terms of the contract executed by the agents or agencies who are acting on behalf of the Advertiser. Conditions set in this document (except the rates) are subject to change by the Publisher without notice. No conditions other than those set forth in this "General Conditions Document" shall be binding upon the Publisher. Advertisers (and their agencies) are fully responsible for the content of their advertisements printed in ***Java Developer's Journal***. Advertisements are to be printed at the discretion of the Publisher. This discretion includes the positioning of the advertisement, except for "preferred positions" described in the rate table. Cancellations and changes to advertisements must be made in writing before the closing date. "Publisher" in this "General Conditions Document" refers to SYS-CON Publications, Inc.

This index is provided as an additional service to our readers. The publisher does not assume any liability for errors or omissions.

Ringing in the New Year

Onno Kluyt

The year 2006 was a great year for community technology development across the board. At the JCP, Spec Leads, Expert Groups members, observers and Executive Committee members worked together to take Java standards to the next level of development. Women Spec Leads had an outstanding contribution; in 2006 several of them won the distinction of Star Spec Leads for their leadership in driving Java specifications from concept, submission, standard development, to Technology Compatibility Kit (TCK) and Reference Implementation (RI) delivery. Ekaterina Chtcherbina was one of them. Always passionate about Java and the community, she felt strongly that "Java technology for me is not just a programming language. Rather it is a new style of technology innovation. Java technology is not created somewhere and given as a final technology to everyone. Instead, the evolution of Java technology relies highly on the community input."

Another woman Star Spec Lead of 2006 was Jaana Majakangas, a senior design engineer at Nokia Corporation. Jaana always found the spec lead work deeply satisfying. "I like my current role where we create a standard in some area, and it is interesting to see how it is then implemented in actual products. This gives us feedback on how we have succeeded," she said.

Linda DeMichiel, senior engineer at Sun Microsystems, also gained the distinction of Star Spec Lead in 2006. Linda provided strong leadership for the Expert Group of JSR 220 (EJB 3.0) and lent her expertise to the effort of making EJB components easier to use and the EJB programming model more flexible and powerful.

Pia Niemela, another woman Spec Lead from Nokia Corporation, earned the Star Spec Lead distinction for her JSR leadership especially for JSR 256 Mobile Sensor API, which she brought successfully to its final development stage.

Six more new spec leads reached stardom in 2006: Danny Coward, Pierre Gauthier, Éamonn McManus, Antti Rantalahti, Bill Shannon, and Shai Gotlib; they consistently set the bar higher with quality and timely delivery of their JSRs. If you want to find out more about these Spec Leads and the JSRs they lead, visit http://jcp.org/en/press/news/star.

More JCP landmarks and successes were recognized at the 2006 JavaOne Conference. The community was brought into the spotlight repeatedly and credited for its role and contributions by leading names in the industry including Sun Microsystems' CEO Jonathan Schwartz and other conference speakers who urged attendees to join the community. A good measure of the 2006 JCP accomplishments is provided by the 4th JCP Annual Awards and its winners. This year, after the nominations round, there were about

three to four candidates for the winner title in each category – all very strong. There are five categories in which contenders vie each year to make the top four or five; this year's winners are noted in parentheses: Member of the Year (Sony Ericsson), Most Outstanding Spec Lead for Java Standard Edition/Enterprise Edition (Linda DeMichiel), Most Outstanding Spec Lead for Java Micro Edition (Asko Komsi, Mark Duesner), Most Innovative JSR for Java Standard Edition/Enterprise Edition (JSR 292 Supporting Dynamically Typed Languages on the Java Platform), and Most Innovative JSR for Java Micro Edition (JSR 272 Mobile Broadcast Service API for Handheld Terminals). Go to http://jcp.org/en/press/news/awards/2006award_nominees for a complete list of the nominees and a description of the awards categories including jurying criteria.

JCP inspired JSR itineraries at the 2006 JavaOne Conference took attendees on interesting journeys of standards discovery. They included sessions regarding key directions of the Java Platform, Standard Edition 6 (Java SE 6), and Java Platform, Enterprise Edition 5 (*Java EE* 5); JSR 270, the topic of an advanced how-to session that presented the scripting features in Java SE 6, including the scripting APIs and the JavaScript ScriptEngine included in the latest release; JSR 224, Java API for XML-Based Web Services (JAX-WS) 2.0; JSR 286, Version 2.0 of the Portlet Specification; JSR 277, Java Module System; JSR 220, *Enterprise JavaBeans (EJB)* 3; JSR, 269, Pluggable Annotation Processing API; JSR 235, Service Data Objects (SDO); JSR 235, Service Data Objects (SDO); JSR 256, Mobile Sensor API; JSR 248, Mobile Service Architecture; JSR 232, Mobile Operational Management; JSR 257, Contactless Communication API; and JSR 272, Mobile Broadcast Service API for Handheld Terminals. These are just a few examples of JSR-based technology events at the 2006 JavaOne Conference. For a complete search of JSR-based sessions, go to http://java.sun.com/javaone/sf/sessions.jsp.

2006 was also the year the JCP Training Program went virtual. With this the JCP has come to the rescue of those who cannot travel to in-person training events and prefer taking the training program virtually through the JCP.org site.

It was also the year of a new effort initiated to improve and change the Java Community Process through JSR 306. From the developer feedback we received regarding the content of the JSR, "Improving involvement of individuals" was the top pick, closely followed by "Optimizing duration of JSRs." Also "Easing migration of existing technologies into standards" got a good number of votes. Summarizing, JSR 306 will explore the possibilities of implementing certain specifications outside the Java platform. In EC talk, these have been labeled as "Hybrid JSRs." By "non-Java," the JSR drivers mean anything that is written and runs entirely outside the Java environment. It could be written in C, in Ruby, in COBOL, or Prolog for that matter. The point is that there are situations where it makes sense to enable the JCP to specify APIs that can be implemented in a Java application and in other architectures. Web services interoperability can be one context; Java language features

–continued on page 62

Onno Kluyt is the director of the JCP Program at Sun Microsystems and Chair of the JCP.

onno@jcp.org

Visit the *New*

www.SYS-CON.com

Website Today!

The World's Leading *i*-Technology News and Information Source

24/7

FREE NEWSLETTERS
Stay ahead of the i-Technology curve with
E-mail updates on what's happening in your industry

SYS-CON.TV
Watch video of breaking news, interviews with industry leaders, and how-to tutorials

BLOG-N-PLAY!
Read web logs from the movers and shakers or create your own blog to be read by millions

WEBCAST
Streaming video on today's i-Technology news, events, and webinars

EDUCATION
The world's leading online i-Technology university

RESEARCH
i-Technology data "and" analysis for business decision-makers

MAGAZINES
View the current issue and past archives of your favorite i-Technology journal

INTERNATIONAL SITES
Get all the news and information happening in other countries worldwide

JUMP TO THE LEADING i-TECHNOLOGY WEBSITES:

IT Solutions Guide
Information Storage+Security Journal
JDJ
Web Services Journal
.NET Developer's Journal
LinuxWorld Magazine
Linux Business News
Eclipse Developer's Journal
MX Developer's Journal
ColdFusion Developer's Journal
XML Journal
Wireless Business & Technology
Symbian Developer's Journal
WebSphere Journal
WLDJ
PowerBuilder Developer's Journal

> From a developer's point of view, it is difficult to understand why public access is not granted on Java specification efforts that the developer is interested in"

–continued from page 60

clearly not. "Hybrid" then describes a JSR that allows both: it still must do all the known Java work and may then also allow the gathered IP to be implemented in another world other than Java.

Increasing openness and transparency of expert group discussions and other related communications at the JCP are also among the goals of the JSR. They are often-raised topics and very valid ones. From a developer's point of view, it is difficult to understand why public access is not granted on Java specification efforts that the developer is interested in. In earlier versions of the JCP, the first draft review, then known as Community Review, was restricted to the JCP membership. Meanwhile we made all draft reviews public and rightly so! Nothing scary happened. Since JCP 2.5, the spec lead and expert group have had considerable freedom over how they conduct their work. Several spec leads have taken that freedom to run their JSRs in a very open manner, with Doug Lea's JSR 166 often used as the prime example. Again nothing scary happened. Many external standards organizations and many JSRs have a desire to work together (OSGi, OMG with CORBA, OMA, and various Java ME–related JSRs are some of the examples). On previous occasions when we looked at this, the solutions always seemed complex. Now, in JSR 306, it appears we may be able to build such liaison relationships and provide that much-sought-after transparency with the same edit to the JSPA, the membership agreement.

Individual developer's participation will be included in the scope of JSR 306 as well. Joining the JCP as an individual is evidently possible as proven by the 700 or so individual members out of the total membership number of 1100, but admittedly it is not a turnkey effort. When the JCP first started in December 1998 it was aimed at enabling corporations and institutions to come together over the standardization of Java technology. The membership agreement might seem lengthy to some; it is because it needs to capture all the IP aspects due to the JCP's mandate that JSRs deliver a spec but also two pieces of software (the RI and TCK). This makes the membership agreement complex to a degree that you're not typically used to dealing with as an individual. The Web site (JCP.org) can also play a role here. In parallel to JSR 306, my team will be working to improve the information provided on the site about the membership process. Keep your opinions coming regarding JSR 306; you can send them to the Spec Lead (onno@jcp.org) or to the JSR expert group (jsr-306-comments@jcp.org).

The JCP wrapped up the year with Executive Committee elections and key JSRs that crossed the finish line. Election congratulations go this year to IBM; Oracle; HP; Fujitsu; Doug Lea, professor of computer science; Motorola; Vodafone; Siemens; BenQ; Ericsson AB; and Jean-Marie Dautelle, individual developer and initiator of several open source projects. They were all elected or re-elected on the JCP SE/EE EC and ME EC, respectively. More details about the EC members and their Java technology and community expertise are posted on jcp.org at http://jcp.org/en/press/news/ec-feature_SE091206 for SE/EE EC and at http://jcp.org/en/press/news/ec-feature_ME091206 for ME EC.

One of the JSRs finalized in the last weeks of December 2006 is JSR 270, Java SE 6 Release Contents, which published its Final Release on December 11. The major themes of this release are compatibility and stability; diagnosability, monitoring and management; ease of development; enterprise desktop; XML and Web services; and transparency. Most of these are continuations of successful themes from the Java SE 5 release. The last theme, transparency, is new and reflects Sun's ongoing effort to evolve the Java SE platform in a more open and transparent manner. As the lead of this umbrella JSR, Sun worked closely with an Expert Group of 18 members including ASF, BEA, Capgemini, Google, HP, IBM, Intel, MetaSolv Software, Oracle, Red Hat Middleware, SAP, SAS, Thoughtworks, and several passionate individual developers and representatives of the academic world. Feedback from the broader Java community was also included in this release through the input developers shared via java.net. The specification including the RI and the TCK can be downloaded from http://jcp.org/aboutJava/communityprocess/final/jsr270/index.html. For a perspective on the top 10 reasons why developers should upgrade to this new release of Java SE, visit Danny Coward's blog at http://blogs.sun.com/dannycoward/entry/java_se_6_top_ten.

Among the component JSRs developed in close synchronization with JSR 270, Java SE 6, which also posted their Final Releases on December 11, are JSR 199, Java Compiler API; JSR 202, Java Class File Specification Update; JSR 221, JDBC 4.0 API Specification; JSR 223, Scripting for the Java Platform; JSR 268, Java Smart Card I/O API; and JSR 269, Pluggable Annotation Processing API.

The other umbrella JSR that published its Final Release just before the turn of the year was JSR 248, Mobile Service Architecture. The main focus of this JSR has been to create a very predictive Java platform for mobile devices through continual architectural consistency, focus, and direction to the collection of efforts for Java ME. It has also been driven by the need in the marketplace for a clear statement on how the various technologies fit and work together. The co-Spec Leads from Nokia and Vodafone note on the JSR Public Page (http://jcp.org/en/jsr/detail?id=248): "This JSR provides guidelines to integrate Java ME JSRs in a uniform and predictable arrangement that is customized specifically for the high-volume handsets. It issues clarifications on certain components when necessary and aims at reducing the number of available options." The Final Release can be downloaded from http://jcp.org/en/jsr/stage?listBy=final.

This is just a quick recap of what went on at the JCP in 2006 – a busy year with important developments and accomplishments that bode well for the new year ahead of us.

Stay tuned for updates, which we will continue to bring to you year round.

Complex and evolving systems are hard to test...

Parasoft helps you code smarter and test faster.

Start improving quality and accelerating delivery with these products:

Awarded **"Best SOA Testing Tool"** by Sys-Con Media Readers	**InfoWorld's 2006 Technology of the Year pick** for automated Java unit testing.	Automated unit testing and code analysis for C/C++ quality.	Memory errors? Corruptions? Leaks? Buffer overflows? Turn to…	Easier Microsoft .NET testing by auto-generating test cases, harnesses & stubs	Automate Web testing and analysis.
SOAtest™	*Jtest*™	*C++test*™	*Insure++*™	*.TEST*™	*WebKing*™

PARASOFT®

We make software work.™

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