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

JDJ

JDJ.SYS-CON.COM VOL.11 ISSUE:11

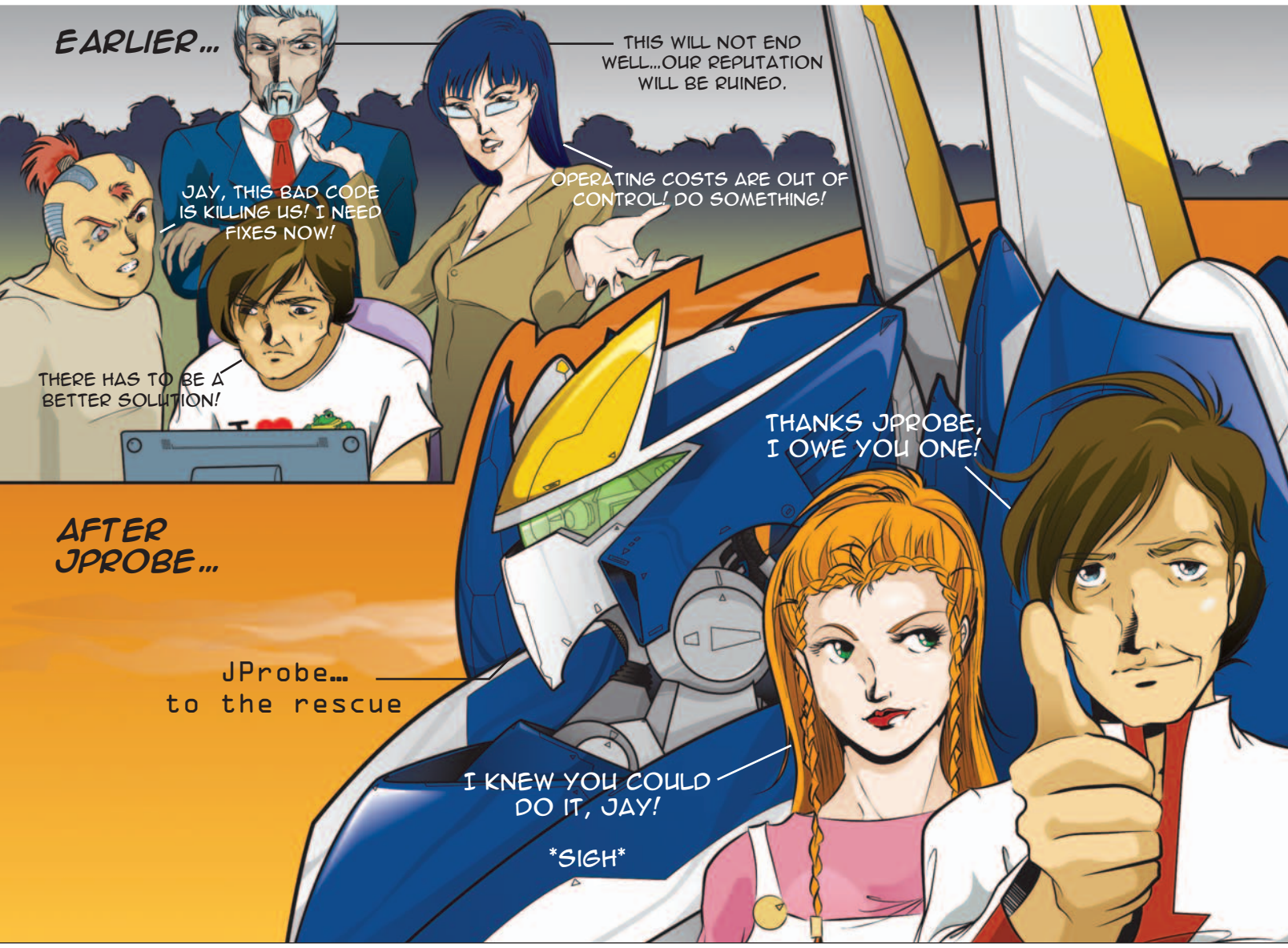
XSLT Solution for Java EE Applications

Adding pervasive computing support to existing applications

RETAILERS PLEASE DISPLAY
UNTIL JANUARY 31, 2007
$5.99US $6.99CAN
12>
0 09281 01751 6

PLUS...

- JDBC 4.0
- Real SOA
- Designing JUnit Test Cases



* Free t-shirt offer

In the Battle Against Bad Java Code…

JProbe® is on your side.

Jay, our Development Manager, was in trouble. He had the Application Business Owner, Production IT Manager and even the CIO on his back about bad code finding its way into production. Operation costs were skyrocketing, productivity was down and customer loyalty was at risk.

Then Jay discovered JProbe® from Quest Software. With JProbe, Jay's team can proactively zero in on the code that affects performance – before the code goes to production. Now Jay is a hero and delivers quality, high-performing code on time – every time.

Join the battle against bad Java code in production. Download a trial of JProbe at www.quest.com/hero — and get a free JProbe t-shirt!*

QUEST SOFTWARE®

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

Is This the Advent of *the Post-Modern Internet*?

Jeremy Geelan

The question that forms the title of this editorial was recently asked by a young observer of the Web 2.0 scene, Skinner Layne. Possessed of a supple mind suffused with historical sense and insight, he contends that the key thing to determine about Web 2.0 is whether it is best characterized as a revolution in Web development or as a rebellion against Web 1.0 – two quite different things.

Layne's chosen analogy is with the French versus the American revolution:

"Web 2.0 can take two distinct directions … [it] can be the French Revolution of Technology or it can be the American Revolution of Technology."

His sense appears to be that Web 2.0 is more of a rebellion, a corrective to Web 1.0, which he calls "a destination-driven experience, one created not by users, but for users, and with little input or insight from them at all."

There is a reason that this interpretation is bad news for Web 2.0 fanboys. As Layne puts it:

"The problem with successful rebellions is that rebels rarely know how to govern or else they take up the mantle of those against whom they rebelled, and like Orwell's pigs in Animal Farm, they begin to sleep in the old rulers' beds."

Indeed Layne's not altogether comfortable with the version number approach in and of itself:

"Web 2.0, Search 2.0, Life 2.0, World 2.0. The metaphor of software versions to describe technological and social phenomena once upon a time was clever. But, as with all clever sayings, it became overused and is now cliché. The draw toward terms like 'Web 2.0' is of course that it makes a strong implication that what it represents is a 'next generation' of something good enough to have gotten a second run. The trouble with such monikers, though, is their post-modern tendency to merely be what came after."

Having introduced the notion of post-modernity into his essay, Layne then drops another word-bomb by referring to "the advent of *the Post-Modern Internet* embodied in the Web 2.0 movement." Thus begging the question: Is "Web 2.0" the Advent of the Post-Modern Internet? [My emphasis.]

There's 10 times more disagreement about what "post-modern" connotes than about what "Web 2.0" means simply because the former term has been around a lot longer than the latter. But even so, it is intriguing to contemplate that a phenomenon as young as the Internet might have already moved into its second era.

Are we entering a new historical period of the Internet and the Web, or merely an extension of the existing one?

"There's 10 times more disagreement about what 'post-modern' connotes than about what 'Web 2.0' means simply because the former term has been around a lot longer than the latter"

Jeremy Geelan is group publisher of SYS-CON Media and is responsible for the development of new titles and technology portals for the firm. He regularly represents SYS-CON at conferences and trade shows, speaking to technology audiences both in North America and overseas.

jeremy@sys-con.com

Gear up for XML excellence

ALTOVA® XML Suite 2007

Take off with the Altova® XML Suite, and save ½ off the top tools for XML development.

Included with the Altova XML Suite 2007:

- Altova XMLSpy®, MapForce®, and StyleVision® Enterprise or Professional Editions
- Plus Altova SchemaAgent™, SemanticWorks™, and DiffDog® with Enterprise Edition
- Also get a FREE copy of Altova DatabaseSpy™ 2007 for a limited time*

The Altova XML Suite 2007 delivers the latest releases of world's leading XML development tools all in an unrivaled deal. It contains Altova XMLSpy, the industry standard XML development environment; MapForce, the premier data integration and Web services implementation tool; and StyleVision, the ultimate visual stylesheet designer. What's more, the Enterprise Edition also includes XML Schema management, Semantic Web, and XML-aware differencing tools. Save a bundle!

Download the Altova XML Suite today: www.altova.com

ALTOVA®

*__Special offer:__ Now until Dec. 24, 2006, purchase or upgrade to the Altova XML Suite and get the NEW Altova DatabaseSpy database query and design tool for FREE!

Conditions apply, see Web site for details.

NOVEMBER 2006 VOLUME:11 ISSUE:11

JDJ contents

JDJ Cover Story

Adding pervasive computing support to existing applications

by Boris Minkin **18**

Features

38

JDBC 4.0
by John Goodson and Mark Biamonte

50

Real SOA
by Andrew Borley, Simon Laws, and Haleh Mahbod

JDJ (ISSN#1087-6944) is published monthly (12 times a year) for $69.99 by SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645. Periodicals postage rates are paid at Montvale, NJ 07645 and additional mailing offices. Postmaster: Send address changes to: JDJ, SYS-CON Publications, Inc., 577 Chestnut Ridge Road, Woodcliff Lake, NJ 07677.

Yakov Fain
Enterprise Editor

Spring + Hibernate EJB3, **POJO + JDBC?**

In the beginning there was nothing: no Java and no data.

Then someone said, let there be data and relational databases with SQL were born.

And someone said, let Java talk to databases, and JDBC was born.

And someone saw that JDBC was good, but someone else saw that JDBC was bad, and EJB with CMP were created.

And someone said, J2EE containers are bad and POJO has resurrected.

And entity beans were slow and heavy; Hibernate was born and people forgot SQL, which was a sin.

And someone said, J2EE is no good, and he divided Spring framework from J2EE.

And fifty more people said nothing is good, and they created fifty more Java frameworks. And poor Java Joe said, "I'm sick and tired of this variety. I'm going back to Java EE."

Some enterprise Java shops that were using J2EE application servers and EJB 2.x found that the combination was overkill for most of their applications, and decided to look for an alternative. Spring framework combined with Hibernate seems to be a logical alternative to J2EE, but will this combo deliver a light weight replacement for Java EE, especially when greatly simplified EJB 3.0 is available?

In my opinion, not only the Spring/Hibernate combo, but even each one separately, is pretty heavy as any framework. Only reusable loosely coupled components are lightweights.

Spring framework is presented as a set of components that can be used separately, but you can also wire them together by adding two pounds of XML. But the minute you do this, you fall into an XML trap. If you use any single component of the Spring framework, it's lightweight. But since it takes two to tango, it's as if you're pulling a tiny roll of thin wire out of your pocket (a.k.a. XML), which becomes heavyweight because wires tend to twist and create a mess.

Concerning Hibernate, I'm not even sure why so many people are using it in the first place. I could see an enterprise architect wanting to use it to lay out a brand new design of a stack of business applications, and to enforce it to a firm-wide standard for data persistence.

–continued on page 10

> "Spring is probably one of the best Java frameworks available today. It has only one drawback: it's a framework"

Yakov Fain is a Managing Principal of Farata Systems. He's authored several Java books and dozens of technical articles. SYS-CON Books will be releasing his latest book, "Rich Internet Applications with Adobe Flex and Java: Secrets of the Masters" this fall. Sun Microsystems nominated and awarded Yakov with the title Java Champion. He leads the Princeton Java Users Group. Yakov teaches Java and Flex 2 at New York University. He is an Adobe Certified Flex Instructor.

yfain@faratasystems.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*
Robyn Forma *robyn@sys-con.com*

Advertising Sales Manager:
Megan Mussa *megan@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*

Associate Editor:
Lauren Genovesi *laureng@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*
Paula Zagari *paula@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*

Rational®

_INFRASTRUCTURE LOG

_DAY 15: This project's out of control. The development team's trying to write apps supporting a service oriented architecture, but it's taking forever. Gil's resorted to giving them all coffee IVs. Now they're on java while using JAVA. Oh, the irony.

_DAY 16: Big crisis—we've just run out of half-and-half!!

_DAY 18: I've found a better way: IBM Rational. It's a modular software development platform based on Eclipse that helps the team model, assemble, deploy and manage service oriented architecture projects. The whole process is simpler and faster, and all our apps are flexible and reusable. The software we write today will be the software we use tomorrow. :)

_The team says it's nice to taste coffee again, but actually drinking it is sooo inefficient!

Download the IBM Software Architect Kit at:
IBM.COM/**TAKEBACKCONTROL**/FLEXIBLE

> Over the last three to four years, many people have been bashing EJBs as an unnecessary complicated framework with lots of convoluted XML descriptors"

–*continued from page 6*

But if you're developing a typical CRUD application, especially when it comes to using already existing and not perfectly designed databases, why even bother with Hibernate? Does SQL scare you that much?

Take an application built on Spring components interconnected with thin wires, put Hibernate on top of it with wires of a different diameter, and the maintainability of your application will decrease while hard-to-find bugs make themselves at home in your application.

Over the last three to four years, many people have been-bashing EJBs as an unnecessary complicated framework with lots of convoluted XML descriptors. Now EJB 3.0, with its annotations, is trying to appeal to enterprise developers again. This won't be easy, because bad memories last for years. But don't kid yourself when you substitute EJB for the Spring/Hibernate combo: it won't make your life much easier.

I do believe in standalone POJOs that know nothing about the environment they're in, but do know how to perform a specific function (i.e., send a message, manage transactions, create a pretty report based on provided SQL, model some financial process, find an optimal route, and the like). Just pass the required parameters to this black box, get the result back, and do whatever you want with it. Inversion of Control or the Dependency Injection paradigm is nothing new, and it works fine. For ten years, I've been routinely using it (without knowing its future name) in my PowerBuilder applications. It was a period of event-driven programming. We were creating user objects with custom events. Whoever wanted to pass some information to this object would fire a custom event that would carry required data and inject them right into the object. Look, ma! No wires! Today, I do the same thing in ActionScript 3. Stop wiring, just write the code required by your business application and forget about it when the new project starts. But don't forget about independent reusable components.

Spring is probably one of the best Java frameworks available today. It has only one drawback: it's a framework.

Hibernate offers you a caching object? Great! Let's use it, without the need to install the whole shebang. Get the caching component somewhere, roll up your sleeves, and create an instance of this object passing all required parameters to its constructor. Stop wiring; get back to programming. The combination of good knowledge of SQL, JDBC, caching (only if needed), and a pagination component (only if needed) can get you pretty far.

At one of my recent presentations to Java developers, I asked the question, "Who knows how to delete duplicates from a database table?" No one knew. When I asked the same question on one of the online forums, some Java developer proudly announced that with Hibernate, you don't create duplicates in the first place. Thank you very much! How about some real world experience? What if the database table with dirty data already exists and dirty feeds keep coming in nightly? Do not kid yourself. Learn SQL.

If you want to write a simple application, don't start by looking for a "light-weight" third or fourth party framework. Program your business logic in POJOs, and your database access in DAOs. Keep it simple. Need transactions? Find a transaction manager. Need scalability? Consider using asynchronous messaging between components.

Floyd Marinescu starts his foreword to the book "Beginning EJB 3" (aPress) as follows:

EJB 3 is a very important milestone for the specification. Not only is it significantly easier to use, but also for the first time (in my opinion), the specification is now built around the proven needs of the development community, standardizing existing best practices instead of being the result of design by committee.

It's great that the bad guys from some evil committee were finally overthrown by the good guys, who are actually paying attention and incorporating best practices and ideas of the multitude of open source frameworks.

And someone said, go back to Java EE standards. And he created Java EE 1.5 and it was good. It was not the best, but it gave people a common ground and fertile soil for seeds of a new generation of enterprise Java applications. Amen.

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.

Jumpstart SOA

Pragmatic approaches to integrating .NET and Java components within WebSphere portal

by Laurence Moroney

The struggle to integrate business assets across the .NET - J2EE technology divide is legendary. So it should come as no surprise that the emergence of portal applications as an enabler of Service Oriented Architecture is forcing enterprises to revisit interoperability challenges in a user-centric environment.

The aim of a portal is to integrate all enterprise data and applications into a coherent whole, and it is wasteful if the portal becomes simply another silo, limited in what it can aggregate due to back end technology constraints. For the portal to be an effective collaboration and productivity enhancement tool, it needs to be populated with enterprise content, irrespective of whether the applications were written in C#, Visual Basic.NET, or Java. This article will survey various technology options that are available for integrating content on the .NET platform into IBM WebSphere Portal Server.

Laurence Moroney is a senior architect and the director of technology evangelism for Mainsoft Corporation, where he is responsible for counseling customers about their interoperability and porting challenges. Previously, he worked in several fields, designing mixed architectures for financial services systems, airports, casinos, and professional sports.

ljpm@sportstalkny.com

Portals as Enablers of SOA: Options for Integration

Consider the scenario where a number of business-critical applications are running on Microsoft's .NET Framework-based technology. The business already has WebSphere Portal running portlets that offer common sign-on and branding. End users want assets that are currently implemented in .NET on the portal, and IT management has the job of figuring out how to make this work.

Rewrite .NET Apps in Java

One approach is to rewrite .NET applications in Java, which is a good strategy if 1) the intention is to migrate all development and infrastructure to Java; 2) the applications are small and there aren't very many of them; and 3) ample Java developers are on hand.

If the .NET application is written using tiers, with a typical nTier architecture offering resource, service, business logic, and presentation tiers – a partial rewrite may be an option. One retailer I recently encountered favored this approach, rewriting the presentation tier as Java portlets, and then consuming their .NET middleware using SOAP Web services. Following this approach, the retailer could achieve its end user requirements such as single sign-on using WebSphere Portal's end-user facilities. A partial rewrite provides all the benefits of a pure Java portlet implementation at the cost of disposing some of the existing work and skills. However, the retailer would face the risk of potential interoperability problems between the .NET and Java Stacks across SOAP Web Services. As the original architecture was .NET end-to-end, it is highly likely that used complex data types such as the .NET DataSet, which interoperate nicely on a .NET stack, will fail on a mixed stack, and as such may necessitate considerable re-engineering of the middleware tiers.

Web Services for Remote Portlets

A second option for a multi-platform scenario is to use a relatively new standard called "Web Services for Remote Portlets" (WSRP). This is a powerful yet limited solution. The concept is simple: Web services typically only communicate data, and not the presentation. With WSRP, the standards-based technology of Web services is expanded

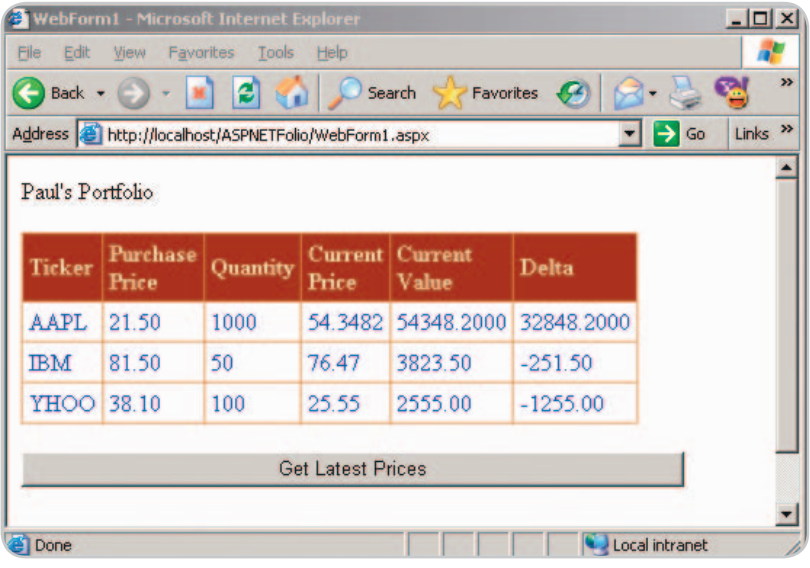
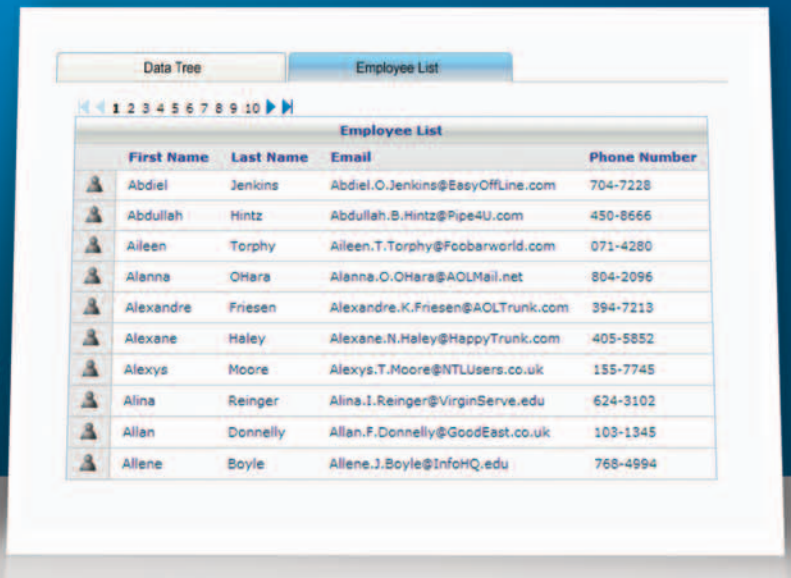


Figure 1 Running the portfolio on IIS

Speed.
Simplicity.
Style.

Build a better User Interface with NetAdvantage



NetAdvantage® for JSF

2006 Volume 1

AJAX-enabled JavaServer™ Faces components

Speed – Built to support large, data-driven applications

Simplicity – Intuitive features for end users; simple tag interface for developers

Style – Fully customizable look & feel

learn more: infragistics.com/jsf

Infragistics Sales - 800 231 8588
Infragistics Europe Sales - +44 (0) 800 298 9055

Infragistics®
Powering The Presentation Layer

WINDOWS FORMS ASP.NET WPF JSF

grids navigation menus listbars trees tabs explorer bars editors

> "...it is wasteful if the portal becomes simply another silo, limited in what it can aggregate due to back end technology constraints"

to deliver HTML markup in the payload. This allows for applications to run on their native platform as WSRP "Producers." Portals that want to integrate them to make WSRP calls to them to get their UI as WSRP "Consumers." It's possible to make modifications to applications that run on the .NET Framework, and expose them as WSRP "Producers," which could then be consumed by a Java portlet running on WPS. However, Microsoft has not yet produced a facility for applications to be exposed as WSRP producers in either its SharePoint product suite or its upcoming one-stop API for integrated applications called the "Windows Communication Foundation." Nevertheless, WSRP can be used to integrate non JSR-168 applications into portals. While WSRP largely preserves the existing .NET development model and the Windows runtime, it does require the amendment of the presentation layer to make it a WSRP producer.

One of the limitations of WSRP is the brittleness it introduces into the environment. If an application is consumed using WSRP from a portal server that requires a sign-on, a portlet that runs on the portal server needs to be written that can read from its credential store, and those credentials are then used to sign onto the WSRP producer using a WSRP call. It would then have to parse the SOAP message containing the WSRP code, including the UI and generate the UI from that, overriding the existing WSRP functionality on the portal server.

Re-hosting ASP.NET Applications as JSR-168 Portlets

A third option for .NET-Java integration – available exclusively for WebSphere Portal – is Mainsoft's cross-platform software, Visual MainWin for J2EE, Portal Edition. This technology recompiles .NET applications to run as JSR-168 portlets that can run natively on WebSphere Portal. The approach delivers a tight integration between .NET and J2EE on the Portal container, providing all the advantages of Java-written portlets, such as single sign on, universal branding, inter-portlet communications, and easy management, without having to reengineer existing .NET components.

Visual MainWin is similar to WSRP in that it preserves existing .NET components and development models. The primary difference is that the runtime is WebSphere rather than Windows. Visual MainWin works by using a patent-pending technology that cross compiles the Microsoft Intermediate Language (MSIL) code generated by the .NET Framework compilers into Java Byte code, and it provides a Java-based implementation of the .NET Framework runtime support classes on which the application will execute. The Portal Edition targets WebSphere Portal, allowing .NET developers to port ASP.NET Web forms applications, ASP.NET Web Services, or .NET class libraries into JSR-168 Portlets, J2EE-based Web services, and J2EE class libraries, respectively. Some reworking is typically necessary to address the paradigm shift between a Web application and a Portlet, but in the majority of cases, this is simply removing hard-coded visual styles and replacing them with ones that the portal server uses to brand portlets running within it.

Visual MainWin also includes a plug into the Visual Studio .NET IDE and enables C# and Visual Basic.NET developers to continue using Microsoft's popular IDE to develop and maintain ASP.NET applications running on WebSphere Portal. The cross-platform tool enables code sharing between development teams using .NET and Java technologies, including the facility to directly consume Java-based assets such as WebSphere Portal APIs and EJBs from within the Visual Studio.NET development environment.

Take, for example, a label on an ASP.NET Web forms application that has been developed using Visual Studio, with font and size attributes that are appropriate for its native environment. In a portal environment, the label's attributes are set by the portal container rather than by Visual Studio so the label is consistent with all other applications in the Portal. When re-hosting the ASP.NET Web forms application, .NET developers remove the code used to set the font type and size properties. This does not suffer from the architectural brittleness of the WSRP approach. Whenever

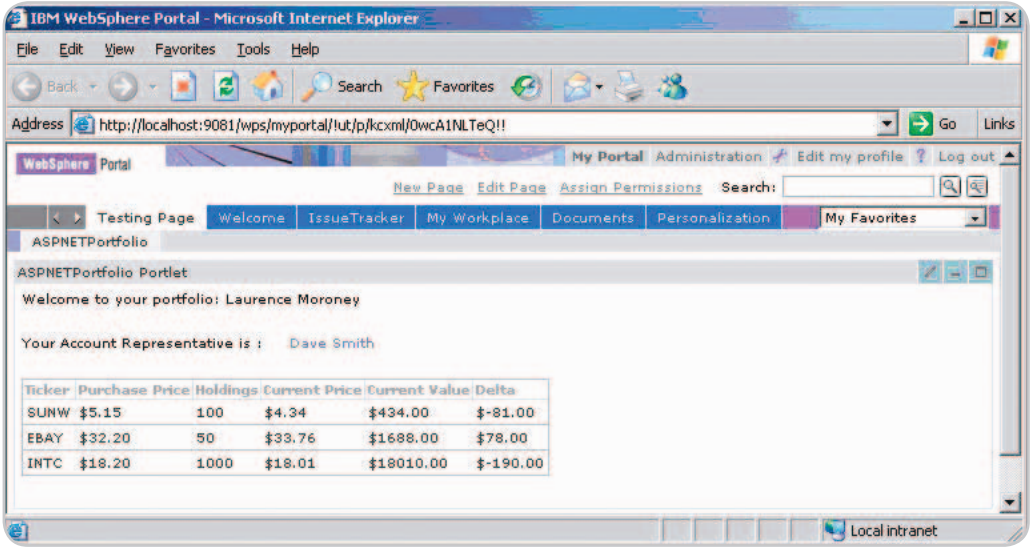


Figure 2 Running the portfolio on WPS

MAKE ANSWERS TO PERFORMANCE PROBLEMS COME TO YOU.

OPNET Panorama offers powerful analytics for rapid troubleshooting of complex J2EE/.NET 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 most successful organizations in the world rely on OPNET's advanced analytics for networks, servers, and applications.

OPNET Panorama™
Real-Time Application Analytics

www.opnet.com/pinpoint

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

	Integration with WPS	Integration with Other Portlets	Manageability	Coding Effort	Preservation of .NET Skills
Full Rewrite in JSR-168	Excellent	Excellent	Excellent	Extensive	Poor
Partial Rewrite in JSR-168	Excellent	Excellent	Good	Moderate	Good
Use WSRP	Poor	Poor	Difficult	Moderate	Excellent
Use Visual MainWin	Excellent	Excellent	Excellent	Minor	Excellent

the styles used to brand the application change, the portal container sends them to the ASP.NET application.

The ability for Visual Studio developers to retain control over the enterprise applications was a primary reason that a Fortune 100 personal insurer pursued this approach. This insurance provider resulted from two insurance providers that merged. One of the providers was built on IBM technologies and the other was a Microsoft shop. Post merger, the insurer had over 1,000 Visual Studio developers, many of whom have been supporting a homegrown Windows portal comprised of about 700 ASP/VB6 applications. Mainsoft and Prolifics, a premier IBM business partner, demonstrated the ability to replace a Windows-based portal with a WebSphere Application Server and WebSphere Portal. The team required five staff-days to migrate a sample ASP/VB6 application to WebSphere Portal.

Sample ASP.NET Portlet Implementation on WebSphere Portal

Take a simple application written as an ASP.NET Web forms application that provides a user interface layer onto middleware implemented using Web services. In this case, the UI layer provides the portfolio holdings. The application is a single Web form that contains a data grid. This data grid uses ADO.NET data binding to an XML data source to present the information. The data source is built from an XML file (in the real world, it would be a middleware service that wraps a database) and a public Web service that provides delayed stock quotes. The overall presentation is defined by hand coding it on the Web forms level. Sign-on is custom written using Windows forms authentication. You can see the application, running on Windows with IIS in Figure 1.

When porting this application to run on WPS, some minor modifications were needed. Many of these are business-driven as opposed to technology shortfalls. For example, the application used its own custom sign-on instead of integrating with the portal sign-on. Thus the code for custom sign-on should be removed, and replaced with code that integrates with the WPS credential store and user management APIs. This is simpler than it sounds, because Visual MainWin enables the ASP.NET application to consume the Java APIs in the same way any other Java application can handle them. Additionally, the hard-coded visual styling should be removed and replaced with the CSS tags that the WebSphere Portal server defines, so that the application can be branded and styled properly.

Once all this is done, it's simply a matter of recompiling the application into Java code, and deploying it to WebSphere Portal using Visual MainWin. Deployment is automatic to the development server, and a WAR file can be generated for deployment to staging and production servers.

The re-hosted application can be seen in Figure 2. This shows tight integration with WPS, providing a welcome message to the user by retrieving her name from the WPS credential cache as well as demonstrating integration with Java-based assets such as the Workplace client people awareness functionality.

The code for this portlet is available for download here at http://dev.mainsoft.com/Portals/0/Downloads/ASPNETPortfolio.zip

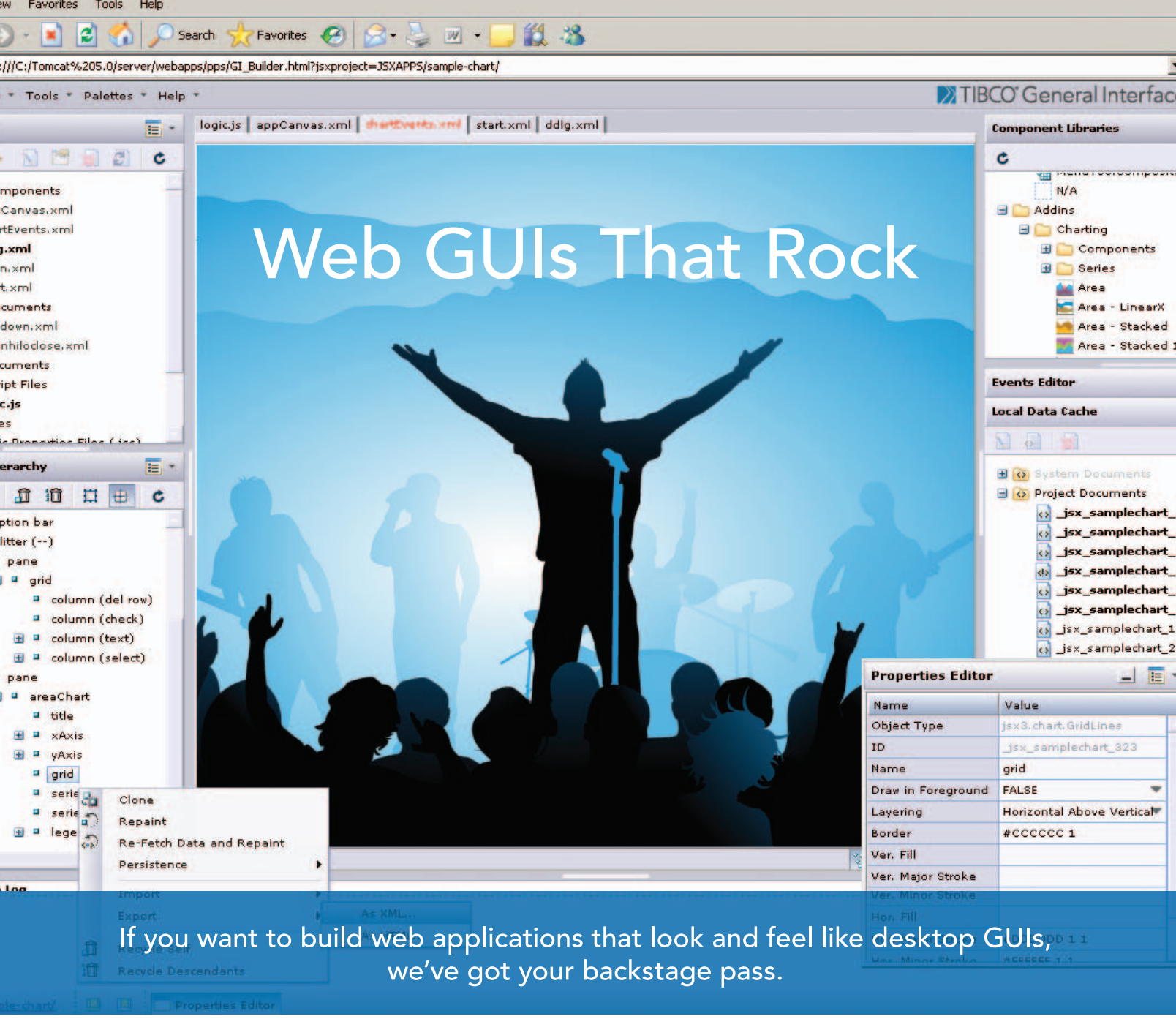
Of course, re-hosting is not limited to existing ASP.NET applications to WPS using this toolkit. Visual MainWin also enables .NET developers to create new applications as Java portlets on WPS that can take advantage of all the underlying functionality for WPS that was previously only available to Java developers (WCT People Awareness, Edit Mode, Portlet-to-Portlet Integration, and "wiring," to name just a few).

Conclusion

Should you have a complex data center where applications run on varied technology stacks, and you want to use the benefits of WebSphere Portal for "on the glass" integration, there are three options available to you:

- Rewrite applications, or their presentation layer, as JSR 168-compliant Java portlets, which can integrate fully into WebSphere Portal. This is ideal in scenarios with limited .NET code in which demand for integrated end user experience is high. It's best suited for a transformation to a Java only shop.
- Use WSRP to consume .NET applications in WebSphere Portal, preserving .NET components and developers. However, this approach has limitations in what it can support. WSRP may require an extensive rewrite of applications to get them to work, and functionalities such as single sign on, and consistent branding can introduce architectural brittleness.
- Visual MainWin for J2EE, Portal Edition runs ASP.NET applications natively on WebSphere Portal, without having to rewrite the application in Java. It also extends WebSphere Portal's rich end user functionalities across C# and Visual Basic components. This approach is useful when implementing a mixed portal environment quickly; when dynamic business requirements exist for.NET applications; and when retention of the current development model is preferred.

Ultimately, the extent of integration requirements, business requirements, and end user needs will define the best solution to use with WebSphere Portal.



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®

OpenAjax alliance MEMBER

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

XSLT Solution for Java EE Applications

Adding pervasive computing support to existing applications

by Boris Minkin

Applying XSLT (eXtensible Stylesheet Language for Transformations) to XML documents can be done using the Java EE (formerly J2EE) Servlet filters model and Java Server Pages (JSP) technology. Servlet filters can be invoked before or after the invocation of a particular servlet or JSP, based on the incoming URL mapping, which could be specified as the central controller servlet in a framework such as Struts or a custom-developed one. The basic logical model is shown in Figure 1.

In Figure 1, a Struts framework is used in combination with Servlet filters technology. When a request comes from the client (Browser, PDA or Smart Phone), the controller servlet of the Struts framework gets called and in turn, invokes an appropriate action based on the mapping of the submitted URL, as defined in Struts configuration file (struts-config.xml). Inside the action, a particular Web controller logic is processed (such as parsing of parameters) and then a business logic bean is invoked (for example, EJB), which communicates with the database server and performs necessary database operations in the appropriate transactional scope. A response from the business bean then is encapsulated in the action response and is delivered to the client by the corresponding JSP.

In this case, the JSP would be coded in the XML form – meaning that the JSP file will not contain HTML as it usually does, but an XML markup instead. A reference to the proper XSLT style sheet would be dynamically generated, based on the MIME type of the incoming request (UserAgent header for most of the HTTP-based clients). Application of XSLT can be done using Servlet Filter technology, where the servlet or JSP response is modified after the processing by the servlet is completed. Modification of the servlet response would encompass the application of the appropriate XSLT style sheet using, for example, Java API for XML Processing (JAXP). This actually simplifies things, since no custom XML parser is required, and one can use standard J2SE technology to utilize all XML-based processing (such as DOM, SAX, and XSLT).

Listing 1 demonstrates the usage of JAXP API for processing and passing transformation to the HTTP Servlet response.

The xmlSource and styleSource variables in the Code Listing 1 have the StreamSource type, containing source of XML and XSLT stylesheets.

In Listing 1, several JAXP classes are used. First, TransformerFactory class follows the Factory design pattern to generate a new instance of XSLT transformers. The file name XSLT would be supplied as a parameter, while original XML source would come from the JSP response. CharArrayWriter is used to buffer the contents to be sent later on to the HttpServletResponse output stream.

Advantages and Disadvantages of This Approach

The advantages of using XSLT include:

- The ability to better separate model, view and controller in the application.
- An additional view tier (XSLT stylesheet) is introduced that is completely separate from any Java or logical processing and encourages developers to use optimal design strategies without mixing Java code with the view (something that commonly happens in the coding of regular JSP pages).
- The JSP is device-independent: it contains just XML, or data which can be converted into any appropriate view for the particular device, be it Wireless Markup Language (WML), XHTML, regular HTML, etc.

The disadvantage of this approach is that we need an extra layer of processing which can introduce additional performance burden on the system. This is because the application of XSLT is generally a processor-intensive operation, and for large JSP pages, especially large style-sheets, performance degradation can result. One way to partially mitigate this problem is to allow the client to handle the application of XSLT. For instance, newer versions of the Internet Explorer browser such as 6.0 SP1 or later include an XSLT engine that's compliant with W3C standards, and can easily replace server based XSLT application. In the client XSLT processing model, the sever will pass XML and XSLT style sheets directly to the browser, and the browser will be responsible for actually applying the stylesheet to the XML data, unlike the server XSLT processing mechanism, when XSLT is applied to XML on the server and the result is passed to the client. However, this approach is limited due to the inability of other client devices to perform similar operations – it would be highly unlikely to expect PDA or Smart Phone to perform such an extensive operation as XSLT's transformation.

Another possibility is that developers experience difficulty debugging XSLT without the use of extra tools. Besides, XSLT is difficult to debug because it is not a complied language. Everything is happening at the runtime during execution of the XSLT application. Industrial-strength development environments,

Boris Minkin is a Senior Technical Architect of a major financial corporation. He has more than 15 years of experience working in various areas of information technology and financial services. Boris is currently pursuing his Masters degree at Stevens Institute of Technology, New Jersey. His professional interests are in the Internet technology, service-oriented architecture, enterprise application architecture, multi-platform distributed applications, and relational database design.

bm@panix.com

such as Microsoft Visual Studio and IBM WebSphere Studio Application Developer include powerful debuggers that help alleviate this issue.

Converting Existing J2EE Application to XSLT Approach

The following few steps will describe the conversion of the existing J2EE applications to use the XSLT approach to display content to multiple device types.

Environment Setup

We'll use DBTestSH application developed in the article entitled: "Bringing Together Eclipse, WTP, Struts, and Hibernate" (http://java.sys-con.com/read/216320.htm). We'll use the following tools and technologies:

- ***J2SE 5.0 JRE***: http://java.sun.com/j2se
- ***Eclipse 3.2***: www.eclipse.org
- ***XSLT Parser***: Xalan-J implementation http://xml.apache.org/xalan-j/
- ***WTP 1.5***: www.eclipse.org/webtools
- ***Tomcat 5.0***: http://jakarta.apache.org/tomcat/
- ***MySQL 4.0.25***: www.mysql.com
- ***MySQL Connector/J driver 3.1***: www.mysql.com/products/connector/j/
- ***Struts 1.1***: http://struts.apache.org
- ***Hibernate 3***: www.hibernate.org
- ***Microsoft Internet Explorer Browser 6.0 & Microsoft Mobile Explorer Emulator***: http://www.devhood.com/ Tools/tool_details.aspx?tool_id=52

The purpose of this exercise is to extend the existing application to use XSLT style sheets to convert our JSP pages (coded as XML documents) into the format appropriate for the particular agent type, e.g., Microsoft Mobile Explorer or Microsoft Internet Explorer.

In order to set up the environment, we need to do a few steps over those described in previous article http://java.sys-con.com/read/216320.htm. First, make a copy of the DBTestSH project (obtained from the previous article). Simply right-click on it and select "Copy." Then right-click again, select "Paste," and the simple screen shown in Figure 2 will be displayed by Web Tools.

After that, several important libraries need to be added under WEB-INF\lib. These are XSLT parser libraries. We'll use open-source Apache Xalan-J processor to do the job; it works quite well with Sun standard JAXP API. The following JARs have to be placed under WEB-INF\lib:

```
xalan.jar          -  Xalan XSLT processor.
xercesImpl.jar     -  implementation of Xerces XML parser.
xml-apis.jar       -  XML APIs for JAXP compatibility.
serializer.jar     - XML serialization APIs.
```

We'll also create a new directory structure under Web Content for our XSLT style-sheets:

- WebContent
  - stylesheets
    - ***ie***: Will contain style sheets for Internet Explorer.
    - ***Wml***: Will contain style sheets for Microsoft Mobile Explorer.

This can be easily done using the Eclipse "New Folder" wizard.

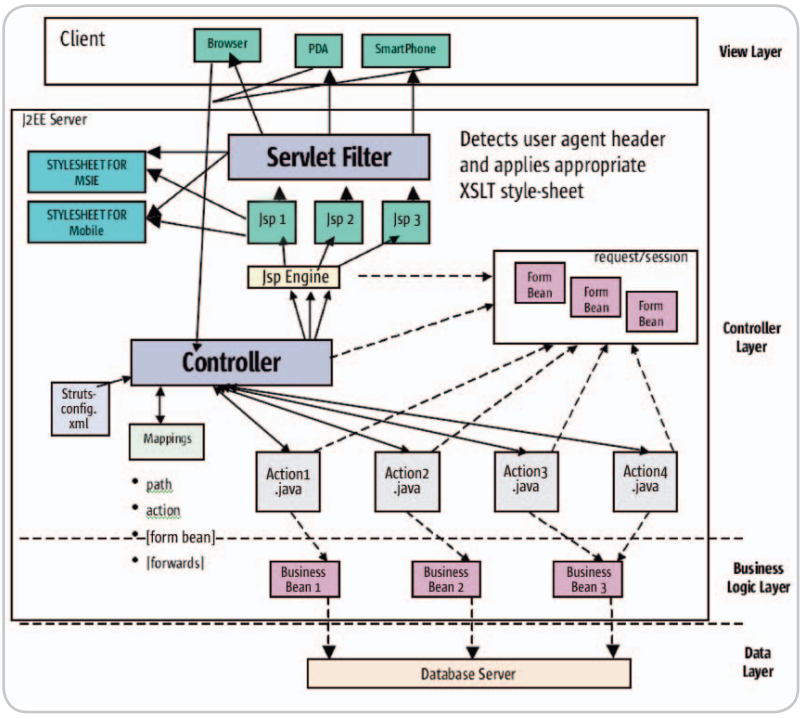


Figure 1 Architecture of the J2EE / Struts application with pervasive support

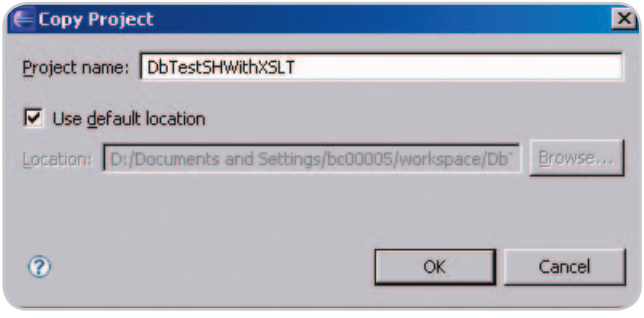


Figure 2 Copy/Paste the project in Web Tools

Servlet Filter Code and Job Description

The conversion job as shown in Figure 1 will be done by a Servlet Filter. Servlet Filters have been a standard part of Servlet API since version 2.3. They allow modifying HTTP response and can serve many different purposes, including enforcing security or access restrictions of the application, caching the application content, etc. In our example, however, we'll use servlet filters to process applications of the XSLT style sheet to the XML data passed from the JSP. My example has been partially borrowed from Sun Microsystems' reference to servlet filters implementation at http://java.sun.com/products/servlet/Filters.html, however, a few modifications were made to the code in Listing 2 in order to fit into our application. For instance, I've added the code to determine the content type based on the "user-agent" header (containing client information, such as browser type) and to determine the appropriate XSLT style sheet based on the provided request URI.

In Listing 2, we use a reference to the CharResponseWrapper class that is used to wrap HttpServletResponse, so its String contents can be easily retrieved when needed.

Listing 3 shows a simple wrapper for HttpServletResponse from a servlet.

Listing 2 has several important methods that were added to fit Sun's example into our application:

- ***determineStylesheetNameFromRequest***: Determines the style sheet name based on the request URI. Another way of doing this is by using XML file mapping, which looks like a more elegant approach, but for the purposes of simplicity, we have just hard-coded the mappings between URIs and corresponding XSLT stylesheets.
- ***determineStylesheetPathFromRequest***: We'll have a couple of directories under our Web Content root directory where XSLT stylesheets will be placed – each directory corresponds to the set of stylesheets for a particular browser type.
- ***determineContentTypeFrom***: We also need to determine the content type of the request. For the Internet Explorer, it would simply be "text/html". For a Wireless Access Protocol (WAP) browser, such as Microsoft Mobile Explorer, or PDA, or a smart phone capable of serving WML content, it would be "text/vnd.wap.wml". Other types can be added by simple modification of this method.

Now, that we covered auxiliary private methods of ApplyXSLTFilter.java, let's talk about how the method doFilter(ServletRequest, ServletResponse, FilterChain) works. Once the content type, the XSLT style sheet name and the path are determined by calling methods at the beginning of the filter processing, we set the response type to the content type determined and get the real OS path for the stylesheet using a handy method called getRealPath(String) of javax.servlet.ServletContext. Now, we have our style sheet source by reading a file using StreamSource class from JAXP.

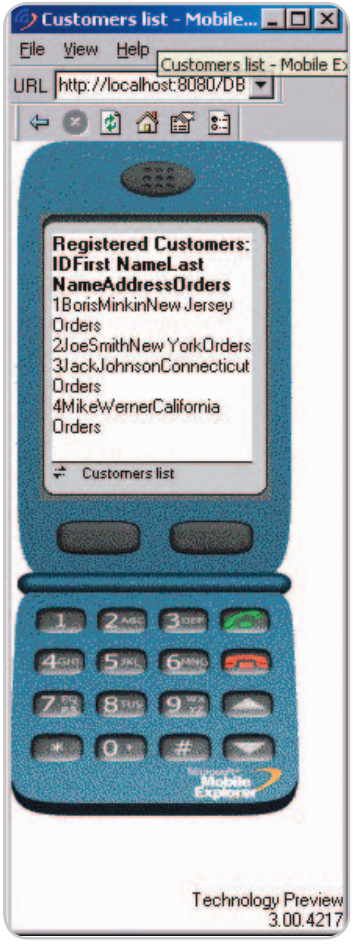


Figure 3 Displaying the customer page on the Mobile Explorer Emulator

In the next step, we obtain the response writer from HttpServletResponse, wrap it in CharResponseWrapper and pass control to doFilter(ServletRequest, ServletResponse, FilterChain), which executes the underlying Struts action and populates our response with data. Obtaining this data is easy if you use the same wrapper class, and we have our XML source – wrapped as StreamSource class from JAXP.

Finally, XSLT transformation will take place and write the transformed output to the HTTP response, displaying it to the appropriate device type.

Setting up the Servlet Filter in the Web Deployment Descriptor

A Java Web application actually knows that it has a servlet filter from the Web Deployment Descriptor: web.xml file, as shown in Listing 4.

This mapping specifies the name of the Servlet filter and its class. IT also defines the association with URI patterns that this filter will be invoked upon. Since we use Struts, *.do pattern will be associated with all the filter invocations. JSP pages can be invoked independently from Struts, so we also need to make sure that the JSP association with Servlet Filter is held on. For that purpose, one minor step needs to be done: index.html page should be renamed to index.jsp, so that our filter knows to handle it. Otherwise, we'd have to add a third mapping for *.html files.

Converting Our JSP Pages

As described at the beginning of this article, existing JSP pages will be split into JSP files producing XML and style sheets with XSLT content. All the scriplets, expressions, and other JSP artifacts will remain in JSP pages, but all the graphics, HTML content, JavaScript, and all the view artifacts will go to the appropriate XSLT style sheets. Listing 5 shows the example of such a conversion.

Now, we remove all the graphics from customer.jsp, all the view producing code (remember MVC?), keeping only the data in XML format, as shown in Listing 6.

As we can see from Listing 6, all the scriplets, expressions, jsp:useBean tags – all this is remaining in the JSP file, but all the view attributes are gone. Where? They go into XSLT stylesheets – one for each browser type – under the corresponding directory, as shown in Listing 7.

For those, who are not familiar with XSLT, it may be somewhat hard to understand the code in Listing 7. Basically, the entire HTML is wrapped into a sing XSLT template that is applied to the root of the XML document during transformation. "xsl:for-each" is a loop construct in XSLT that allows looping through a bunch of XML data – in our case customer data. At every step, it displays the corresponding customer data using "xsl:value-of select" construct that displays a particular attribute of the CUSTOMER element from our JSP/XML file (such as ID, first name, last name, etc.). Note the "xsl:output" element, which designates the content type as text/html.

Listing 8 is the style sheet for WML output.

When we invoke the Mobile Explorer Emulator, we should be able to see our customer list on a mobile phone screen, as shown in Figure 3.

Conclusion

In this article, I showed you how to convert a Java Web application to be used with multiple device types. It added an extra layer of processing, but at the same time increased application flexibility greatly by allowing supporting multiple independent views using the same back-end model, in the letter and spirit of Model-View-Controller paradigm. Using Eclipse and Web Tools simplifies this job even further, as they provide JSP editors, XML editors, and validators that allow one to make sure their XSLT and XML source is valid and will execute properly.

My wish list for Web Tools starts with a XSLT debugger that would allow debugging XSLT processing just like in Java. This would help tremendously in figuring out what's wrong with your XSLT code, but for now, we still have to rely mainly on the console messages.

–Listings on page 22

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

Listing 1: Example of JAXP XSLT Transform to HTTP Servlet Response

```
TransformerFactory transformerFactory =
     TransformerFactory.newInstance();
Transformer transformer =
transformerFactory.newTransformer(styleSource);
CharArrayWriter caw = new CharArrayWriter();
StreamResult result = new StreamResult(caw);
transformer.transform(xmlSource, result);

response.setContentLength(caw.toString().length());
out.write(caw.toString());
```

Listing 2: ApplyXSLTFilter.java

```
package filter;

import java.io.CharArrayWriter;
import java.io.File;
import java.io.IOException;
import java.io.PrintWriter;
import java.io.Reader;
import java.io.StringReader;

import javax.servlet.Filter;
import javax.servlet.FilterChain;
import javax.servlet.FilterConfig;
import javax.servlet.ServletException;
import javax.servlet.ServletRequest;
import javax.servlet.ServletResponse;
import javax.servlet.http.HttpServletRequest;
import javax.servlet.http.HttpServletResponse;
import javax.xml.transform.Source;
import javax.xml.transform.Transformer;
import javax.xml.transform.TransformerFactory;
import javax.xml.transform.stream.StreamResult;
import javax.xml.transform.stream.StreamSource;

public class ApplyXSLTFilter implements Filter {

 TransformerFactory transformerFactory = TransformerFactory.newIn-
stance();
 FilterConfig filterConfig = null;

 /**
  * Perform filtering action to apply XSLT
  */
 public void doFilter(ServletRequest request, ServletResponse
response,
    FilterChain chain) throws IOException, ServletException {

  HttpServletRequest req = (HttpServletRequest)request;

  System.out.println("Starting the filter...");
  System.out.println("Path info: " + req.getRequestURI());

  // Determine XSLT attributes
  String styleSheetPath = determineStylesheetPathFromRequest(req);
  String contentType = determineContentTypeFrom(req);
  String styleSheetName = determineStylesheetNameFromRequest(req);
  String styleSheet = styleSheetPath + styleSheetName;

  System.out.println("Here is the stylesheet we are going to load:
" + styleSheet);

  response.setContentType(contentType);
  String stylePath =
                this.filterConfig.getServletContext().
getRealPath(styleSheet);
  File styleFile = new File(stylePath);

  if (!styleFile.exists()) {
      System.out.println ("File " + styleFile + " does not exist");
  } else {
      System.out.println ("File " + styleFile + " exists!");
  }

  System.out.println("Retrieving stylesheet Real Path: " + style-
Path);
  Source styleSource = new StreamSource(styleFile);

  PrintWriter out = response.getWriter();
  CharResponseWrapper wrapper = new CharResponseWrapper((HttpServlet
Response)response);
  chain.doFilter(request, wrapper);

  // Get response from Servlet
  StringReader sr = new StringReader(wrapper.toString());
  Source xmlSource = new StreamSource((Reader)sr);

  // Ok... Transform the xml:
  try {
   Transformer transformer = transformerFactory.newTransformer(style
Source);
    CharArrayWriter caw = new CharArrayWriter();
    StreamResult result = new StreamResult(caw);
    transformer.transform(xmlSource, result);

    response.setContentLength(caw.toString().length());
    out.write(caw.toString());
  } catch (Exception ex) {
    out.println(ex.toString());
    out.write(wrapper.toString());
  }

 }

 public void init(FilterConfig arg0) throws ServletException {
  this.filterConfig = arg0;
 }

public void destroy()  {
 this.filterConfig = null;
}

 /**
  * Determine stylesheet path from request
  * @param req
  * @return
  */
private String determineStylesheetPathFromRequest(HttpServletRequest
req) {

  String userAgent = req.getHeader("user-agent");
  if (userAgent.indexOf("MSIE") > -1) {
```

```
    return "/stylesheets/ie/";
   }


   if (userAgent.indexOf("MobileExplorer") > -1) {
    return "/stylesheets/wml/";
   }

   return "/stylesheets/ie/";
  }

  /**
   * Determine stylesheet name from request
   * @param req
   * @return
   */
private String determineStylesheetNameFromRequest(HttpServletReque
st req) {

  String uri = req.getRequestURI();
  int index = uri.lastIndexOf('/');
  uri = uri.substring(index + 1);
  System.out.println("Parsed URI: " + uri);

  if (uri.equals("index.jsp")) return "index.xsl";
  if (uri.equals("ListCustomers.do")) return "customers.xsl";
  if (uri.equals("customers.jsp")) return "customers.xsl";
  if (uri.equals("CreateCustomer.do")) return "customer_created.
xsl";
  if (uri.equals("customer_created.jsp")) return "customer_created.
xsl";
  if (uri.equals("CreateOrder.do")) return "order_created.xsl";
  if (uri.equals("order_created.jsp")) return "order_created.xsl";
  if (uri.equals("ListCustomerOrders.do")) return "orders.xsl";
  if (uri.equals("orders.jsp")) return "orders.xsl";
  if (uri.equals("failure.jsp")) return "failure.xsl";

  return null;

}

  /**
   * Determine content type request
   * @param req
   * @return
   */
  private String determineContentTypeFrom(HttpServletRequest req) {

   String userAgent = req.getHeader("user-agent");
   if (userAgent.indexOf("MSIE") > -1) {
    return "text/html";
   }

   if (userAgent.indexOf("MobileExplorer") > -1) {
    return "text/vnd.wap.wml";
   }

   return "text/html";
  }

}
```

Listing 3: CharResponseWrapper.java

In the code above, we use reference to CharResponseWrapper class that is used to wrap HttpServletResponse, so its String contents can be easily retrieved when needed.

```
package filter;

import java.io.CharArrayWriter;
import java.io.PrintWriter;

import javax.servlet.http.HttpServletResponse;
import javax.servlet.http.HttpServletResponseWrapper;

public class CharResponseWrapper extends HttpServletResponseWrapper
{

 private CharArrayWriter output;

 public String toString() {
  return output.toString();
 }

 public CharResponseWrapper(HttpServletResponse response) {
  super(response);
  output = new CharArrayWriter();
 }

 public PrintWriter getWriter() {
  return new PrintWriter(output);
 }
}
```

Listing 4: Filter Mapping

```
<filter>
  <filter-name>ApplyXSLTFilter</filter-name>
  <filter-class>filter.ApplyXSLTFilter</filter-class>
 </filter>
 <filter-mapping>
  <filter-name>ApplyXSLTFilter</filter-name>
  <url-pattern>*.do</url-pattern>
 </filter-mapping>
 <filter-mapping>
  <filter-name>ApplyXSLTFilter</filter-name>
  <url-pattern>*.jsp</url-pattern>
 </filter-mapping>
```

Listing 5: The original customers.jsp

```
<%@ page language="java" contentType="text/html; charset=ISO-8859-
1"
    pageEncoding="ISO-8859-1"%>
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN">
<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-
8859-1">
<title>Customers list</title>
</head>
<body>

<jsp:useBean id="customers" type="java.util.ArrayList<domain.
```

```
Customer>" scope="request"/>

<b>Registered Customers:</b><br>
<table border="1">
<tr>
 <th>ID</th>
 <th>First Name</th>
 <th>Last Name</th>
 <th>Address</th>
 <th>Orders</th>
</tr>
<% for(domain.Customer c : customers) { %>
<tr>
 <td><%= c.getId() %></td>
 <td><%= c.getFirstName() %></td>
 <td><%= c.getLastName() %></td>
 <td><%= c.getAddress() %></td>
 <td><a href="/DBTestSH/ListCustomerOrders.do?cust_id=<%= c.getId()
%>">Orders</a></td>
<% } %>

</body>
</html>
```

Listing 6: XML-based customers.jsp

```
<?xml version="1.0" encoding="UTF-8"?>

<jsp:useBean id="customers" type="java.util.ArrayList<domain.
Customer>" scope="request"/>

<DATA_LIST>
<% for(domain.Customer c : customers) { %>
<CUSTOMER
 ID="<%= c.getId() %>"
 FIRST_NAME="<%= c.getFirstName() %>"
 LAST_NAME="<%= c.getLastName() %>"
 ADDRESS="<%= c.getAddress() %>"
 LINK="/DBTestSHWithXSLT/ListCustomerOrders.do?cust_id=<%=
c.getId() %>"
/>
<% } %>
</DATA_LIST>
```

Listing 7: XSLT Style sheet for the customer display in the Microsoft Internet Explorer: customers.xsl

```
<?xml version="1.0"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
version="1.0">

<xsl:output method="html" media-type="text/html"/>

<xsl:template match="/">

<html>
<head>
<meta http-equiv="Content-Type" content="text/html; charset=ISO-
8859-1"/>
<title>Customers list</title>
</head>
<body>

<b>Registered Customers:</b><br/>
<table border="1">
<tr>
 <th>ID</th>
 <th>First Name</th>
 <th>Last Name</th>
 <th>Address</th>
 <th>Orders</th>
</tr>
<xsl:for-each select="DATA_LIST/CUSTOMER">
<tr>
 <td><xsl:value-of select="@ID"/></td>
 <td><xsl:value-of select="@FIRST_NAME"/></td>
 <td><xsl:value-of select="@LAST_NAME"/></td>
 <td><xsl:value-of select="@ADDRESS"/></td>
 <td><a>Orders<xsl:attribute name="href"><xsl:value-of
select="@LINK"/></xsl:attribute></a></td>
</tr>
</xsl:for-each>
</table>

</body>
</html>

</xsl:template>
</xsl:stylesheet>
```

Listing 8: WML style sheet for customer display

```
<?xml version="1.0"?><xsl:stylesheet xmlns:xsl="http://www.
w3.org/1999/XSL/Transform" version="1.0">

<xsl:output method="wml" media-type="text/vnd.wap.wml"/>
<xsl:template match="/">
<wml>

<card title="Registered Customers">
<table columns="5">
<tr>
 <th>ID</th>
 <th>First Name</th>
 <th>Last Name</th>
 <th>Address</th>
 <th>Orders</th>
</tr>
<xsl:for-each select="DATA_LIST/CUSTOMER">
<tr>
 <td><xsl:value-of select="@ID"/></td>
 <td><xsl:value-of select="@FIRST_NAME"/></td>
 <td><xsl:value-of select="@LAST_NAME"/></td>
 <td><xsl:value-of select="@ADDRESS"/></td>
 <td><a>Orders<xsl:attribute name="href"><xsl:value-of
select="@LINK"/></xsl:attribute></a></td>
</tr>
</xsl:for-each>
</table>
</card>
</wml>

</xsl:template>
</xsl:stylesheet>


....
```

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

Creating a Flashy Monitoring Application

by Yakov Fain

Building an application in Flex using declarative GUI language MXML mixed with ActionScript 3 and XML

Do you know what's the main goal of any gas station owner? To get lots of trucking accounts. Business from small car drivers is worth pennies, and it gets on my nerves to hear them ask again and again, "Five dollars of regular, please." Trucks are different. They usually pump in a couple of hundreds of gallons at a time. For instance, here comes a flashy 18-wheeler with a sign "Software Delivered." These guys ship reusable open source components around the globe. As a former programmer, I was trying to play it smart by asking why they don't just let people download these components from the Internet? But the smiley truckers (many of whom used to be software developers too) just shrug and tell me that nothing beats personal delivery, plus the tips. If you think about it, their ventures are the basis of a new business model. Ten years ago, only professional vendors would create and sell well-documented, working software. Then, independent developers started writing code jointly, but since they did not bother spending time writing documentation and testing, they gave away their software for free, while charging a premium for adding custom features and explaining undocumented ones. Now there is a new trend: delivering software to your doorsteps for tips. This makes sense to me - most paid programmers live in Bangalore, and since they can't deliver, they can't compete.

As you might have noticed from my previous article (http://java.sys-con.com/read/204697.htm), I value truckers' opinions, and this time I've asked them to recommend something to use for the development of the front end for my Gas Station monitoring application. They suggested Flex 2 from Adobe. I followed their advice, and here's how I built this application in Flex using the declarative GUI language MXML mixed with ActionScript 3 and XML, which is pretty easy to read for any Java programmer.

Yakov Fain is a senior IT architect consulting Wall Street companies. He's authored several Java books, dozens of technical articles and his blog is hugely popular. SYS-CON Books will be releasing his latest book, *Rich Internet Applications with Adobe Flex and Java: Secrets of the Masters* this fall. Sun Microsystems has nominated and awarded Yakov with the title Java Champion. He leads the Princeton Java Users Group. Yakov teaches Java and Flex 2 at New York University.

yfain@sys-con.com

Developing with MXML, XML, and ActionScript

Our application will monitor daily operations: gasoline sales and purchases. The output window of this application is pictured in Figure 1.

We'll read the initial gas station activities data from the XML shown in Listing 1.

The final version of our application will include a timer with a random data generator that will add new messages to the window from Figure 1, emulating the data feed of three types of messages: sale, purchase, and spill. In real life, I'll be using Java and message-oriented middleware on the server, but this is beyond the scope of this article.

The first version of GasStation.mxml (see Listing 2) reads and parses the data from GSActivities.xml using this one liner:

```
<mx:XML id="activities" source="GSactivity.
xml" />
```

No additional XML parsing is required.

Behind the scenes, Flex creates an object with the reference variable activities, which is used as a data provider for the data grid as follows:

```
<mx:DataGrid  id="messageBook" dataProv
ider="{activities.message}" width="100%"
height="100%">
```

The dataProvider activities.message represents the <message> XML element from GSActivity.xml, which is displayed as a row in the data grid. The curly braces mean that our XML object is bindable, and that the content of the XML element <message> from Listing 1 will be used to populate each row of the data grid. The ActiveScript renderer function paid(), that is being called for each row, calculates the amount by multiplying the number of gallons and the price per gallon. The <mx:CurrencyFormatter> ensures that the calculated column "paid" is displayed as a dollar amount.

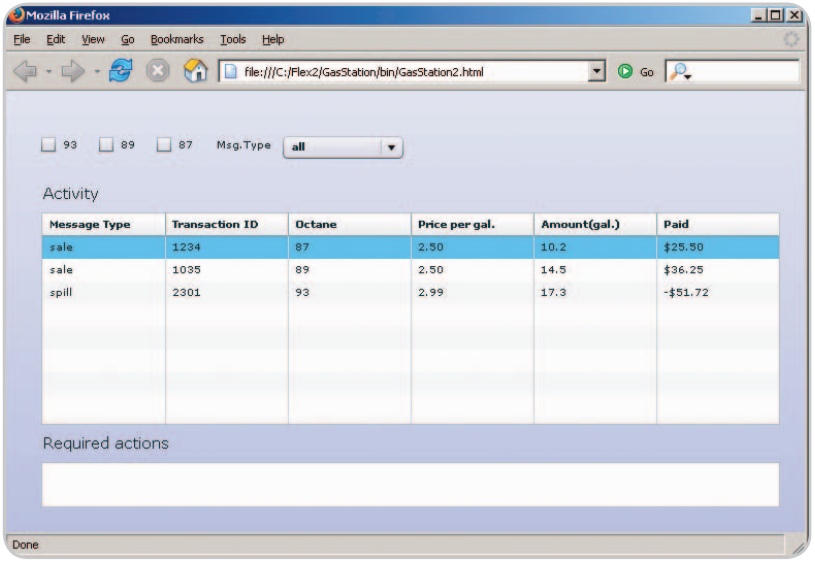


Figure 1 Running GasStation1

The rest of the code in Listing 2 displays other controls that we'll use for filtering and illustrating master-detail relations later in this section.

Note: The combobox cbMsgTypes is populated from an array messageType, which is marked as [Bindable] and will be used later for filtering the messages in the data grid. Also, since in this version we did not define the data grid column, Paid By, the corresponding data from the data provider are not shown.

Adding a Collection Class

Flex collection classes implement Ilist and ICollectionView interfaces, which allow you to add, remove, and update items in a collection. These interfaces also have methods for dispatching events when the data in the underlying collection change, which becomes handy when you use a collection as a data provider of one of the controls. Just add a new element to such a collection and the data in these controls automatically reflect the change.

There are several ways of using collections as data providers, but I'll just show you one.

Eventually, I'll add a middleman between the XML object and the data grid. Now the data grid's provider will become an XMLListCollection built on top of activities XML:

```
<mx:XML id="activities" source="GSactivity.
xml" />
<mx:XMLListCollection id="msgList"
source="{activities.message}" />

<mx:DataGrid  id="messageBook" dataProvider="{m
sgList}">
```

Just recompile and run the application again – it will display the same window as in Figure 1.

Filtering

The next step is to allow the user to filter data by the octane (the check boxes) or the message type (the combo box). Let's add a function init() that will be called on by the applicationComplete event to assign the filter function filterMessages() to the collection, and perform filtering. For Java programmers this line is a bit unusual:

```
msgList.filterFunction=filterMessages;
```

In Flex, you can assign the name of the function to an object's property and execute it. The filtering will happen when we call the function refresh() on the collection.

```
<mx:Application xmlns:mx="http://www.adobe.com/2006/
mxml"
 backgroundColor="#e0e0FF" applicationComplete="ini
t()">
…
   private function init():void {
           // assign the filter function
             msgList.filterFunction=filterMessages;

              // perform filtering
              msgList.refresh();
   }

         private function filterMessages(item:
         Object):Boolean{
             // Check every checkbox and the com
             bobox and
             // populate the datagrid with rows
             that match
             // selected criteria
      if (item.octane=="87"  && this.cbx87.selected)
               return true;
      if (item.octane=="89"  && this.cbx89.selected)
               return true;
      if (item.octane=="93"  && this.cbx93.selected)
               return true;

             return false;
         }
```

Run the application after making these changes, and you'll see an empty table on the screen. This is because, when creation of the application was complete, Flash VM called the method init, which assigned the filter function to our XMLListCollection, and then refresh() applied this filter to the each XML node of our collection. Since no checkbox was selected, the function filterMessages correctly returned false to each node, leaving the datagrid empty. To fix this, let's make a slight change in the checkboxes so they will be initially checked off:

```
   <mx:CheckBox id="cbx93" label="93"
selected="true"/>
   <mx:CheckBox id="cbx89" label="89"
selected="true"/>
   <mx:CheckBox id="cbx87" label="87"
selected="true"/>
```

Now the program will show all the rows again. Try to uncheck the boxes – nothing happens, because the application doesn't know that it needs to re-apply the filter function to the msgList again. This is an easy fix – let's refresh the msgList on each click on the checkbox:

```
<mx:CheckBox id="cbx93" label="93"
selected="true" click="msgList.refresh()"/>
<mx:CheckBox id="cbx89" label="89"
selected="true" click="msgList.refresh()"/>
<mx:CheckBox id="cbx87" label="87"
selected="true" click="msgList.refresh()"/>
```

Filtering by octane number is done. By adding the code snippet below to the beginning of the filterMessages() function, you'll engage filtering by message type according to the combobox selection:

```
if (cbMsgTypes.selectedLabel !="all" &&
                                  item.@msgType!=c
bMsgTypes.selectedLabel ){
   return false;
}
```

Please note the @ sign. This is how you access XML attributes (see Listing 1) using E4X notation. The XML elements are accessed using a regular dot notation.

Master-Details Relationships

The turnover rate at my gas station is pretty high, and I often need to teach new gas attendants how to react to different messages when I'm away. So let's add some help to new employees by populating the Required Actions text area, based on the selected message type. This is a typical master-detail relationship task, where the data grid with messages is "the master," and the text box shows details.

We'll start with creating an actions.xml file where we store recommended actions for each message type (see Listing 3).

To read and parse this file into an XML object, we need to write yet another one liner:

```
<mx:XML id="msgTypes" source="MessageTypes.xml"
/>
```

The next step is to specify that selecting a different row in a datagrid should call the function that finds and displays an appropriate message from MessageTypes.xml. Again, E4X makes this job a breeze:

```
private function getAction():void {

  txtAction.text=
    msgTypes.message.(@type==messageBook.
selectedItem.@msgType).actions;
}
```

> Our application will monitor daily operations: gasoline sales and purchases"

The expression msgTypes.message.(@type==messageBook.selectedItem.@msgType) means the following: select the XML <message> element, which has an attribute type the same as in the selected row in the datagrid in column @msgType. When this XML element is identified, assign its <actions> value to the text area txtAction.

As I stated earlier, changing the selected row in the data grid should initiate the getAction() function call. Let's modify the declaration and add the change event processing:

```
<mx:DataGrid id="messageBook"
dataProvider="{msgList}" width="100%"
                              height="100%"
change="getAction()">
```

If you compile and run this program, then click on a row in a data grid, the action text box will be populated, as shown in Figure 2.

We're almost there. Why almost? Because if the user starts filtering the data by octane or a message type, the actions text field won't be cleaned. To fix this, let's create our own function refreshData() that will not only refresh the XMLListCollection, but also clean the text field:

```
private function refreshData():void{
        msgList.refresh();
        txtAction.text="";
}
```

Don't forget to replace all calls to msgList.refresh() with refreshData().

Adding a Data Feed

In the real world, all the messages would be pushed to our application by some kind of a messaging program. Another possibility is to have the gas station front end program poll the data during specified intervals from a server-side program that can be written as a JSP, ASP, PHP, or whatever else can bake an HTTPResponse. I've shown an example of Flex-JSP communication at http://webddj.sys-con.com/read/264915.htm. For simplicity, we'll emulate a real-time data feed by using a random number-generator and a timer that will add items to our msgList collection at specified time intervals. Since the data collection will be constantly receiving new data, the output window should reflect this by adding new rows into the data grid. Listing 4 has the final code of the application.

This version of the application was built based on the XML data feed, not because it's the best solution for these kinds of applications, but rather to introduce the reader to the ease of XML parsing with E4X. If the speed of your data feed is crucial, or, as a Java programmer you'd like to have more control over the data, use ActionScript objects as a data feed and ArrayCollection instead of XMLListCollection. This object should define getters providing data for all data grid columns, including the calculated amount for the column Paid. Keeping calculations in the function paid() is not a good idea because the so-called label function is called every time the program inserts a new element into the underlying XML collection. Flash repaints each visible data grid row upon each insertion of a new gas transaction, which means the paid amounts for each visible row will be recalculated.

Granted, you can create simple prototype applications in Flex with few lines of code, but let's not fool ourselves - making efficient applications still requires programming, such as in good old Java. To see this application in action, just point your Web browser at http://samples.faratasystems.com/gasstation/GasStation3.html. Adobe Flash Player 9 is required to run this application, but if you don't have it installed, my HTML will automatically detect it - just follow the prompts and it'll be installed pretty quickly.

For a bit more advanced Flex-Java application read the article "Rich Internet Applications with Adobe Flex 2 and Java" at http://java.sys-con.com/read/210991.htm.

In the past, I used only Java for all my programming needs. But as the medieval saying goes, if the only tool you have is a gasoline hose, every problem looks like a car tank. These days are gone, and now I'm using Java on the server-side and Adobe Flex as a tool for building Rich (as in Bill Gates) Internet Applications.

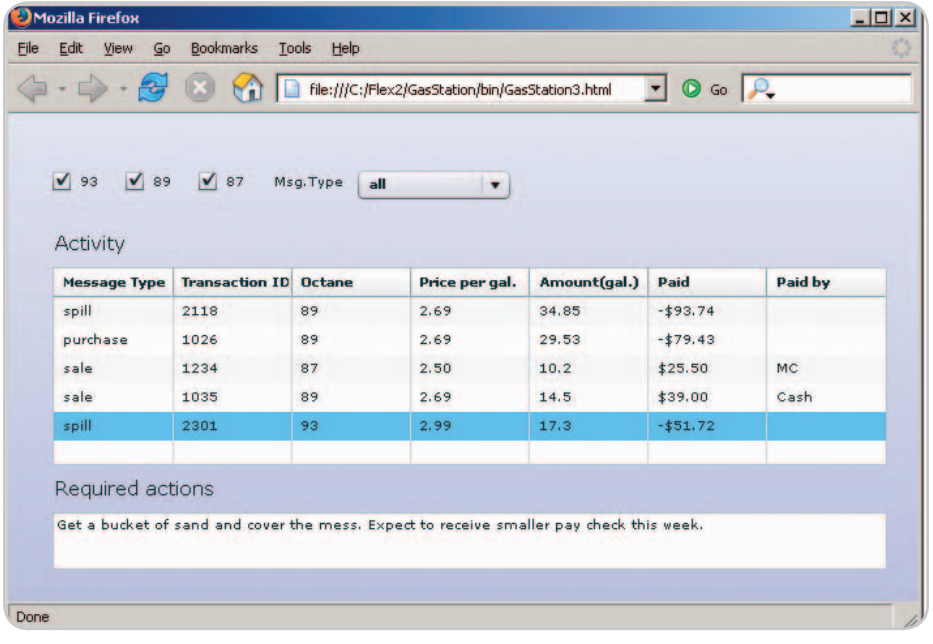


Figure 2 Populating the required actions field

Listing 1: GSActivities.xml

```
<messages>
 <message msgType="sale">
   <transID>1234</transID>
   <octane>87</octane>
   <price>2.50</price>
   <gallons>10.2</gallons>
   <paidby>MC</paidby>
 </message>
 <message msgType="sale">
   <transID>1035</transID>
   <octane>89</octane>
   <price>2.69</price>
   <gallons>14.5</gallons>
   <paidby>Cash</paidby>
 </message>
 <message msgType="spill">
   <transID>2301</transID>
   <octane>93</octane>
   <price>2.99</price>
   <paidby></paidby>
   <gallons>17.3</gallons>
 </message>
</messages>
```

Listing 2: GasStation1.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml" backgroundColo
r="#e0e0FF">

<mx:XML id="activities" source="GSactivity.xml" />
 <mx:Canvas x="10" y="10" width="100%" height="100%">
      <mx:HBox x="10" y="20" width="100%" height="30">
    <mx:CheckBox id="cbx93" label="93"/>
    <mx:CheckBox id="cbx89" label="89"/>
    <mx:CheckBox id="cbx87" label="87"/>
    <mx:Label text="Msg.Type" />
    <mx:ComboBox id="cbMsgTypes" width="117"
                                          dataProvider="{messageType}
"></mx:ComboBox>
   </mx:HBox>
   <mx:VBox x="10" y="64" height="100%" width="100%">
    <mx:Label text="Activity" width="100%" fontSize="15"/>
    <mx:DataGrid  id="messageBook" dataProvider="{activities.message}"
width="100%"
      height="100%">
     <mx:columns>
      <mx:DataGridColumn headerText="Message Type"
dataField="@msgType"/>
 <mx:DataGridColumn headerText="Transaction ID" dataField="transID"/>
      <mx:DataGridColumn headerText="Octane" dataField="octane"/>
      <mx:DataGridColumn headerText="Price per gal." dataField="price"/>
      <mx:DataGridColumn headerText="Amount(gal.)" dataField="gallons"/>
          <mx:DataGridColumn headerText="Paid" labelFunction="paid"/>
     </mx:columns>
    </mx:DataGrid>
    <mx:Label text="Required actions" fontSize="15" />
    <mx:TextArea id="txtAction" width="100%"/>
   </mx:VBox>
 </mx:Canvas>

    <!--Defining USD formatting -->
    <mx:CurrencyFormatter id="usdFormatter" precision="2"
        currencySymbol="$" useThousandsSeparator="false"
alignSymbol="left" />

   <mx:Script>
   <![CDATA[

       //Data for the Message type  combo
         [Bindable]
         private var messageType: Array = ["all","sale", "spill", "pur-
chase"];

   private function paid(item:Object, column:DataGridColumn):String {
    // calculate total gain/loss
        var total:Number=Number(item.gallons)* Number(item.price);
        if (item.@msgType!="sale"){
           total*=-1;
        }
        return ""+usdFormatter.format(total); //USD formatting
   }
   ]]>
 </mx:Script>
</mx:Application>
```

Listing 3: MessageTypes.xml

```
<MessageTypes>
  <message type="sale">


<description>Sale of gasoline products</description>
      <actions>Sale is good news. No action required
    </actions>
  </message>
  <message type="purchase">
    <description>Purchase of gasoline products from suppliers</descrip-
tion>
    <actions>If the gas station owner is not on premises, please call
him at 212-
                  123-4567. Otherwise no actions is required
    </actions>
  </message>
  <message type="spill">
    <description>Spill of gasoline products on the ground</description>
    <actions> Get a bucket of sand and cover the mess. Expect to
receive smaller
              pay check this week.
    </actions>
  </message>
</MessageTypes>
```

Listing 4: GasStation3.mxml

```
<?xml version="1.0" encoding="utf-8"?>
<mx:Application xmlns:mx="http://www.adobe.com/2006/mxml"
 backgroundColor="#e0e0FF" applicationComplete="init()">

<mx:XML id="msgTypes" source="MessageTypes.xml" />
<mx:XML id="activities" source="GSactivity.xml" />
<mx:XMLListCollection id="msgList" source="{activities.message}" />
 <mx:Canvas x="10" y="10" width="100%" height="100%">
      <mx:HBox x="10" y="20" width="100%" height="30">
    <mx:CheckBox id="cbx93" label="93" selected="true"
click="refreshData()"/>
    <mx:CheckBox id="cbx89" label="89" selected="true"
click="refreshData()"/>
    <mx:CheckBox id="cbx87" label="87" selected="true"
click="refreshData()"/>
    <mx:Label text="Msg.Type" />
    <mx:ComboBox id="cbMsgTypes" width="117" dataProvider="{messageType}"
>
              <mx:change>refreshData()</mx:change>
    </mx:ComboBox>
   </mx:HBox>
   <mx:VBox x="10" y="64" height="100%" width="100%">
    <mx:Label text="Activity" width="100%" fontSize="15"/>
```

```
   <mx:DataGrid  id="messageBook" dataProvider="{msgList}" width="100%"
                                          height="100%"
change="getAction()">
     <mx:columns>

 <mx:DataGridColumn headerText="Message Type" dataField="@msgType"/>

      <mx:DataGridColumn headerText="Transaction ID"
dataField="transID"/>

      <mx:DataGridColumn headerText="Octane" dataField="octane"/>
      <mx:DataGridColumn headerText="Price per gal." dataField="price"/>
      <mx:DataGridColumn headerText="Amount(gal.)" dataField="gallons"/>

      <mx:DataGridColumn headerText="Paid" labelFunction="paid"/>
      <mx:DataGridColumn headerText="Paid by" dataField="paidby"/>

     </mx:columns>
   </mx:DataGrid>
   <mx:Label text="Required actions" fontSize="15" />
   <mx:TextArea id="txtAction" width="100%"/>
  </mx:VBox>
 </mx:Canvas>

    <!--Defining USD formatting -->
    <mx:CurrencyFormatter id="usdFormatter" precision="2"
        currencySymbol="$" useThousandsSeparator="false"
alignSymbol="left" />

    <!-- Gallons Amount formating with 2 digits after dec.point -->
    <mx:NumberFormatter id="numberFormatter" precision="2"/>

 <mx:Script>
  <![CDATA[

         //Message type  combo data
        [Bindable]
        private var messageType: Array = ["all","sale", "spill", "pur-
chase"];

        import mx.collections.*;
        private var sortMessages:Sort;

        [Bindable]
        private var grandTotalSale:Number=0;

  private function init():void {
    // assign the filter function
            msgList.filterFunction=filterMessages;

             // perform filtering
             refreshData();

           // emulating message feed in specified intervals
            var myTimer:Timer = new Timer(5000, 0); // every 5 sec
            myTimer.addEventListener("timer", addMessage);
            myTimer.start();

  }

        private function filterMessages(item:Object):Boolean{

            // filter by message types
           if (cbMsgTypes.selectedLabel !="all" &&
                        item.@msgType!=cbMsgTypes.selectedLabel ){
               return false;
          }

            // Check every checkbox and the combobox and
            // populate the datagrid with rows that match
            // selected criteria
          if (item.octane=="87"  && this.cbx87.selected)
             return true;
          if (item.octane=="89"  && this.cbx89.selected)
             return true;
          if (item.octane=="93"  && this.cbx93.selected)
             return true;

            return false;
        }

  private function paid(item:Object, column:DataGridColumn):String {
   // calculate total gain/loss. Label function is not
   // the best place for calculations as it's being called
   // on each change of the underlying collection
       var total:Number=Number(item.gallons)* Number(item.price);
       if (item.@msgType!="sale"){
          total*=-1;
       }

       return ""+usdFormatter.format(total); //USD formatting
  }

  private function getAction():void {
             txtAction.text=msgTypes.message.(@type==messageBook.
selectedItem.@msgType).actions;
  }

  private function refreshData():void{
         msgList.refresh();
         txtAction.text="";
  }

        private function addMessage(event:TimerEvent):void{
        // create and add one message with randomly-generated
        // values to the collection
           var newNode:XML = new XML();
           var transID:String=Math.round(Math.random()*5000).toString();
           var octanes: Array = ["87", "89", "93" ];
           var octaneIndex:Number=Math.round(Math.random()*2);
           var octane:String=octanes[octaneIndex];
           var prices: Array = [2.49, 2.69, 2.99 ];
           var price:Number=prices[octaneIndex];

           var msgTypes: Array = ["sale", "purchase", "spill"];
           var msgType:String=msgTypes[Math.round(Math.random()*2)];


           var payTypes: Array = ["MC", "Visa", "Cash" ];
           var payType:String=msgType=="sale"?payTypes[Math.round(Math.
random()*2)]:"";

           var gals:String=(numberFormatter.format(Math.random()*50).
toString());

          newNode=<message msgType={msgType}>
                    <transID>{transID}</transID>
                    <octane>{octane}</octane>
                    <price>{price}</price>
                    <gallons>{gals}</gallons>
                    <paidby>{payType}</paidby>
                  </message>;

         // adding new messages always on top
         activities.insertChildBefore( activities.message[0], newNode);
        }
  ]]>
 </mx:Script>
</mx:Application>
```

Adobe | Solution PARTNER

CYNERGY

THE LEADERS

IN RIA DEVELOPMENT SERVICES

INCREDIBLE APPLICATIONS
PASSIONATE USERS
PROVEN SUCCESS

Unlock your potential with the help of industry leaders in Rich Internet Application development. 10 Years. 1400+ Customers. **Your app starts here.**

cynergysystems.com

instantiations CASE STUDY

BEA Uses RCP Developer™ to Improve Quality and to Save Time & Money

BEA relies heavily on the WindowTester™ component of the RCP Developer™ software from Instantiations to automate testing of GUI elements

"In terms of our cost of investment in the WindowTester tool, we believe the cost of the licensing is greatly worth the enhancement and efficiency gained so far."
—Bill Roth, VP of BEA Workshop Unit

This Case Study is available online at **www.instantiations.com/rcpdeveloper/resources/casestudy-bea.pdf**

BEA Systems and Instantiations have had a unique relationship while working in the Eclipse ecosystem. BEA is a strategic member of the Eclipse Foundation, has a representative on the Eclipse Board, and is an active participant in the Eclipse Web Tools project. Instantiations is a long time Eclipse member company, has a representative on the Eclipse Foundation Board, and has had a number of committers on Eclipse projects. The work delivered by Instantiations in the Eclipse Pollinate project provided early proof-of-concept technology for BEA's Eclipse-based tooling.

BEA Workshop makes developing Java applications easier by allowing Eclipse developers to quickly create, debug and test SOA components, Web Services, Web applications, BEA WebLogic Portal applications, and enable Service-Oriented Architecture (SOA) solutions. BEA Workshop is based on the Eclipse Open Source Integrated Development Environment (IDE). Eclipse includes a Rich Client Platform (RCP) layer that makes it easy to develop applications with extensive GUI capabilities.

BEA also utilizes other key Eclipse-based tools to develop and test their own applications. For example, BEA replies heavily on the WindowTester component of the RCP Developer software from Instantiations to automate the testing of GUI elements. RCP Developer is a software development product that accelerates the creation of Eclipse RCP applications by providing tools for constructing and testing graphical user interfaces, composing Help documentation and packaging rich client applications for deployment. We will explore how BEA is using the WindowTester component of RCP Developer in the remainder of this case study.

Development Challenge

Three years ago, Workshop's automated IDE-based testing was from a home-grown testing infrastructure. BEA is shifting its product release schedule so releases will be delivered in 1/6th the time previously required. According to Steve Tocco, BEA Workshop Director of Quality Assurance, "Our internal IDE testing had inadequate code coverage numbers with 1/20th the number of tests our current automated test suite contains. We determined that the amount of intermittent failures, the inadequate automated coverage, as well as the cost to implement tests was prohibitive in getting products to market quickly." The BEA Workshop group needed a better solution for automated testing of their application GUIs. They chose Instantiations RCP Developer and its innovative WindowTester functionality to meet this need.

Solution: Using WindowTester for Automated GUI Testing

Testing History

In addition to the in-house testing infrastructure used by BEA Workshop group, they previously used and evaluated other commercial off-the-shelf products. While some of the tools provided a quick-click and record of Swing tests, they would not address the needs raised as Workshop moved to the Eclipse infrastructure in 2004. They researched options and decided to initially create their own test suite using the open source Abbot SWT GUI testing framework. For the product release that shipped in July 06, they used the Abbot structure for testing.

Components of RCP Developer™ 2.0

- **SWT Designer™** automatically generates Java code from a powerful and intuitive visual designer
- **WindowTester™** records events and automatically generates GUI tests based on the JUnit standard
- **Help Composer™** streamlines the creation of documentation that is fully compatible with the Eclipse Help system
- **RCP Packager™** quickly automates deployment by generating a flexible SWT-based Installer

However, the team found that creating and running tests still did not meet their needs. The group decided that developing and maintaining their own test harness was not their core competency and they did not want to devote extensive resources to creating a test infrastructure. Tocco states, "To meet a rapid release schedule, the test suite must be optimized as we can't afford false negatives or instabilities drawing precious resources away from our deliverables. Even more, tests need to be authored rapidly as we try to find optimizations in our schedules while not compromising quality for our customers. We weren't getting that in any form before. Our team felt we weren't agile enough to meet the new release timelines—we simply don't have time now to rely on manual tests alone."

Moving to WindowTester for Automated GUI Testing

The BEA Workshop team began looking at tools that could automate the testing of GUI elements of their Eclipse-based applications. They chose RCP Developer from Instantiations and are converting their test suite to use its WindowTester component. They selected a commercial product even though there are open source alternatives available because of the tools' ease of use as well as Instantiations' support, responsiveness, and professional documentation. According to Bill Roth, VP of BEA Workshop Unit, "In terms of our cost of investment in the WindowTester tool, we believe the cost of the licensing is greatly worth the enhancement and efficiency gained so far."

How Testing Has Changed after Implementing WindowTester

When evaluating RCP Developer and WindowTester, the BEA Workshop for WebLogic team found that it took about two months to achieve a 40% line coverage in the product from a code coverage perspective. According to Tocco, "Since we started using WindowTester, tests that took two to three weeks to write previously can now be done in two to three days. It is much faster to get a test developed and running. In addition, we have a higher stability rate than before with a extraordinary pass rates in our automated regression tests. This is fantastic. It lets us spend our energy building the product, not chasing test issues." The BEA Workshop for WebLogic team also found that WindowTester can handle a heavy test load. The team currently runs over 300 tests. Some of the tests are quite broad, with multi-step lengthy testing scenarios. This is a 20 times increase over what was in the release two years ago for IDE automation.

Future: Full Migration to WindowTester

The BEA Workshop for WebLogic team plans a complete migration away from the old test harness for the IDE in six months. The team intends to use the WindowTester component of RCP Developer as the sole tool for automated IDE testing tool in Eclipse by the end of 2006.

Summary

Moving to Instantiations RCP Developer and its WindowTester functionality has allowed the BEA Workshop for WebLogic group to drastically cut the time it takes to generate new GUI tests. The move to WindowTester has saved the group both time and money allowing them to focus on developing their product rather than creating and maintaining a test infrastructure.

"Since we started using WindowTester, tests that took 2–3 weeks to write previously can now be done in 2–3 days."
—Steve Tocco, BEA Workshop Director of Quality Assurance

About BEA — *www.bea.com*

BEA Systems is a company founded in 1995 that specializes in enterprise infrastructure software, and has 77 offices in 37 countries. BEA Systems, Inc. is a world leader in enterprise infrastructure software, delivering powerful standards-based platforms for building enterprise applications and managing Service-Oriented Architectures even in heterogeneous IT environments. Customers depend on BEA Tuxedo®, WebLogic®, and AquaLogic™ product lines to reduce IT complexity, leverage existing resources, and speed the delivery of new services. BEA also provides support for Blended strategies that combine Open Source and commercial software to best suit the needs of business and IT. With over 15,000 customers including the majority of the Fortune Global 500, BEA provides the technology, solutions and services to help companies achieve a state of Business LiquidITy™ where enterprise assets are freed up to deliver maximum business value.

About Instantiations — *www.instantiations.com*

Instantiations, Inc. provides leading-edge software products, services and technologies for Eclipse, Java and Smalltalk. Instantiations offers professional development environments and software products that integrate seamlessly with the latest development platforms. Instantiations is a member of the Eclipse Foundation and offers a line of products for Eclipse, Rational Application Developer, IBM WebSphere Studio and MyEclipse. Based in Portland, Ore., Instantiations was founded in 1997 by a team of internationally recognized pioneers in the field of component software technology.

About Eclipse — *www.eclipse.org*

Eclipse is an open source community whose projects are focused on providing an extensible development platform and application frameworks for building software. Eclipse is an open source community whose projects are focused on providing a vendor-neutral open development platform and application frameworks for building software. **The Eclipse Foundation** is a not-for-profit corporation formed to advance the creation, evolution, promotion, and support of the Eclipse Platform and to cultivate both an open source community and an ecosystem of complementary products, capabilities, and services.

Designing JUnit Test Cases

Effective functional testing

by Nada daVeiga

Functional testing, or integration testing, is concerned with the entire system, not just small pieces (or units) of code. It involves taking features that have been tested independently, combining them into components, and verifying if they work together as expected. For Java, this testing is typically performed using the JUnit framework.

Most Java developers are well-versed in logistical test construction matters, such as how to develop a test fixture, add test methods with assertions, use the setup method for initialization, and so forth. However, many Java developers could benefit from a deeper understanding of how to develop a functional test suite that effectively verifies whether code works as designed.

This article introduces and demonstrates the following strategy for building an effective JUnit functional test suite:

- Identify use cases that cover all actions that your program should be able to perform.
- Identify the code's entry points – central pieces of code that exercise the functionality that the code as a whole is designed to undertake.
- Pair entry points with the use cases that they implement.
- Create test cases by applying the initialize/work/check procedure.
- Develop runtime event diagrams and use them to facilitate testing.

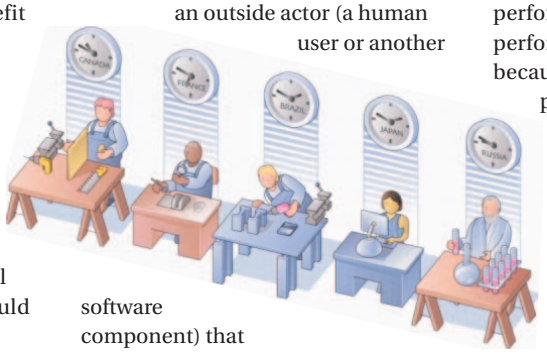
I demonstrate these strategies by applying them to source code from the Saxon project (http://saxon.sourceforge.net/), an XML utility kit that can process XPath, XQuery, and XSLT. This library is built from approximately 50,000 lines of Java code; it is open source, well written, and well documented.

Nada daVeiga is the Product Manager of Java Solutions at Parasoft, where she has been a senior member of the Professional Services team for two years. Nada's background includes development of service-oriented architecture for integration of rich media applications such as Artesia Teams, IBM Content Manager, Stellent Content Server, and Virage Video Logger. Nada developed J2EE enterprise applications and specialized in content transport frameworks using XML, JMS, SOAP, and JWSDP technologies. As a presales engineer, Nada worked with clients such as Cisco, Fidelity, HBO, and Time Warner. Nada holds a bachelors degree in computer science from the University of California, Los Angeles (UCLA).

Identifying Use Cases

There are two balancing goals of functional testing: coverage and granularity. In order to be complete, functional testing must cover each function that the program provides, and it must do so at a level that separates the tests into their component parts. Tests can rely on each other, but no single test should verify two things.

The first step to creating a comprehensive functionality test suite is assembling a list of all the actions that your program should be able to perform. This can be further codified by specifying *use cases* that model a supported action that can be taken by an outside actor (a human user or another software component) that performs work inside the system.



A typical enterprise Java application already has several documents detailing the requirements of the various users. These may include use case specifications, nonfunctional requirements specifications, test case specifications, user interface design documents, mockups, user profiles, and various additional artifacts. Simple applications typically have one simple text document that details all relevant requirements.

Using these documents, you can quickly identify use cases that should be tested. Each test case describes a scenario that can be exercised through the application. A good practice is to aim for similar-sized scenarios that verify one and only one functionality – larger scenarios can be broken into smaller ones along the lines of the functionalities that they verify.

There are many ways to model use cases, but the simplest is in terms of input/output pairs. In Saxon's query class, the simplest use case is passing a query file, a query, and a path to an output file. The output file is created as needed and filled with the result of running the query in the query file.

More complex use cases may take more input or produce more output. The defining point, however, is that use cases do not specify or care how the work is performed internally. They treat the software as a "black box" inside of which all work could be performed by gnomes, as long as it's performed. This is an important point because the use cases as input/output pairs translate very easily and very directly into test cases, which allows complex specifications to map into simple tests that can verify that the required operations work, and that operations which should fail actually fail.

Defining the use cases for the designated entry points is simple if the class is relatively straightforward, or if there is already a specification document that enumerates all of the possible class uses. If not, it might be necessary to learn about the various ways the class is expected to behave (and possibly highlight confusion as to the class's purpose and use). Use cases can also be extracted from the code itself if you are willing to look for all of the places where the code is called.

Most likely, the class has some rudimentary documentation, and by supplementing this documentation with the developers' domain knowledge, it should be possible to fully determine what the class should and shouldn't be able to do. With this knowledge, an appropriate set of use cases can be developed.

> There are many ways to model use cases,
> but the simplest is in terms of input/output pairs"

Translating Test Cases

Each test case is divided into two parts: input and expected output. The input part lists all the test case statements that create variables or assign values to variables. The expected output part indicates the expected results; it shows either assertions or the message 'no exception' (when no assertions exist).

The basic input/output format is the simplest, easiest to understand model to follow for test cases. It follows the pattern of normal functions (pass arguments, get return value), and most user actions (press this button to perform this action). The pattern, then, is to:

- ***Initialize:*** Create the environment that the test expects to run in. The initialization code can either be in the beginning of the test or in the setUp() method.
- ***Work:*** Call the code that is being tested, capturing any interesting output and recording any interesting statistics.
- ***Check:*** Use assert statements to ensure that the code worked as expected.

For instance, assume that you want to test the Saxon library's transform class entry point. One of its use cases is to transform an XML file into an HTML file, given an XSL file that describes the transformation. The inputs are the paths to the three files, and the output is the contents of the HTML file. This translates very directly into the following test:

```
public void testXSLTransformation() {
  /* initialize the variables
    (or do this in setUp if used in many tests) */
  String processMePath = "/path/to/file.xml";
  String stylesheetPath = "/path/to/stylesheet.xsl";
  String outputFilePath = "/path/to/output.xml";
  //do the work
  Transform.main(new String[] {
    processMePath,
    stylesheetPath,
    "-o", outputFilePath } );
  //check the work
  assertTrue(checkOutputFile(outputFilePath));
}
```

Each step can be as simple or complex as necessary. The variables declared here could just as easily call methods to obtain their values. The work could consist of several steps that achieve the desired outcome. Moreover, the check can sometimes be omitted when the process succeeds silently.

The pattern is very simple and very flexible, but step two is decidedly generic. This template gives us no method for finding the code to be tested, or any assurances that the code is set up in a way that facilitates testing. This is a serious concern.

Focusing Functional Tests

The search can be narrowed to a more useful context by identifying central pieces of code that exercise the functionality that the code as a whole is designed to undertake. These classes are considered the code's *entry points* because they provide a way to jump into the system from the outside.

The overall goal of this process is to identify a group of classes that provide a high-level interface to the system functionality. The easier it is to use each class independently, the better. After all, the more the class can be decoupled from its surroundings, the easier it is to test.

Determining what code to identify as entry points is a fairly straightforward process. In a library of code, there are usually a choice few entry points that control all of the library's functionality. These facade classes act as a mediator between client code and the library, separating the developer on the outside from the complexity of the code within. This is exactly the type of class whose methods should be tested first.

For instance, Saxon provides a small collection of classes that act as a portal into the rest of the library, and thus serve as a logical entry point. By coding to the facade classes such as transform, configuration, and query, library client code can use a vast number of worker classes without having to worry about their interfaces… or even their existence. These facade classes therefore provide a simple way to test the system functionality using the high-level and easy-to-use interfaces that are a sign of a good library.

In application code, there is usually an obvious separation between modules of functionality. In some code, these modules are segregated to the extent that they can largely be treated as if they were each separate libraries whose functionality can be accessed through a handful of facade classes. These classes are the logical places to look for high-level interfaces. A plug-in architecture will usually follow this design, in that each individual plug-in has a simple interface that can effectively exercise the entirety of the contained code.

In less rigidly delineated systems, there is usually a central point through which all activity passes. This mediator class is often a 'controller' in an MVC paradigm, and it routes requests to and from parts of the system. The vast majority of the overall system functionality is implemented by classes to which this controller connects; consequently, these classes are prime candidates for testing. This can be seen in Applet design, where the class deriving from java.applet.Applet will be the central processor of the entire code base. Depending on whether the code is thoroughly decomposed, testing can focus on either the Applet subclass itself, or on those classes with which it works.

Code between modules is also prime code to test. The class that converts application requests into database queries is a good candidate, as are similar adapter classes.

Various MVC (Model-View Controller) framework-based components may be easier to test with other testing frameworks, some of which extend JUnit. For example, Struts actions are best tested using the StrutsTestCase extension of JUnit, server-side components like Servlets, JSPs, and EJBs are best tested using Cactus, and HttpUnit is the best framework for conducting black-box Web application testing. All techniques discussed in this article are applicable when writing tests in these frameworks.

Moving from Use Case to Test Case

Once the entry points have been discovered, they must be paired with the use cases that they implement. In some cases, this is a trivial step because the classes' names self-document to the point that matching is automatic: for instance, Saxon's transform class performs the XSL transformations; the query class performs the XQuery resolutions, etc.

In other cases, the search is more difficult. Often, a use case describes functionality that exists only as a cross-cutting concern that is not exemplified in any single class; the behavior in question is visible only when a group of classes interacts, or when certain conditions apply. In these cases, the test has a longer than average initialization phase, or the setUp() method can be used to provide the requisite environment.

The work phase, where the code is actually being called, should be only a single line if possible. Minimizing the contact with the tested code helps you avoid depending on side effects and unstable implementation details.

The test's check phase is commonly the most complex because it must often compensate for code that was not written to be tested. The test may be forced to pull apart the results to ensure that they satisfy the requirements. Occasionally, the results are so difficult to obtain that multiple steps are required to get them into a form that the test can recognize. Both of these cases are true in the above test for XSL transformations; the results are in a file, which must be read into memory, and are in a complex XML format, which must be scrutinized to ensure accuracy.

A simpler example can be taken from Saxon. Given an XML file and an XPath expression, Saxon can evaluate the expression and return a list of all matches. Saxon ships with a sample class – the XPathExample class – that does precisely this. Paring down the interactivity, the class resolves to this test:

```
public void testXPathEvaluation() {
  //initialize
  XPathEvaluator xpe = new XPathEvaluator(
    new SAXSource(new InputSource("/path/to/file.xml")));
  XPathExpression findLine =
    xpe.createExpression("/some/xpath[expression]");
  //work
  List matches = findLine.evaluate();
  //check
  assertTrue(matches.count() > 0);
}
```

The two inputs are the two constant strings, and the output is the list of matches, which is tested to ensure that matches were indeed found. All the work is performed in one line, which simply calls the method that is being tested.

A variation of this case is the expected behavior of XPathEvaluator when the createExpression() method is passed null. In this case, you can expect some error to occur because the expression cannot be created from nothing.

It's a bad idea to keep the input source name and expression in the test case. Some items (server names, usernames and passwords, etc.) should not live in the test files, which should be configurable for the specific deployment. Rather, design the test cases to facilitate separation of test drivers and test data, test driver reuse over a larger set of data, and test data reuse over a larger set of test drivers. On the other hand, don't over-architect a simple test case implementation. Typically, the test case specification already defines most of the system states and allows parameter descriptions for a scenario, so there's no point in making everything parameterizable in the test implementation.

Many code sections may appear in more than one test case. An experienced object-oriented developer will be tempted to re-factor these and create common classes and utility methods. In some cases, that makes a lot of sense – for example, a logging procedure should be a common method available for all test cases. However, be careful not to over-engineer the tests – these Java classes are no more than tests that are meant to validate functional behavior of an application.

The test cases are typically fragile. For example, if a developer changes the location of the input file in the testXPathEvaluation test or if the createExpression method changes its signature, the test scripts will fail. Significant rework and change for each test case implementation is inevitable as an application evolves. Thus, traceability is crucial for all test cases. If something fails later, it's important to be able to point the developer to the corresponding test case specification and use case specification to promote speedy bug resolution. Therefore, test cases should be annotated with references to the original specification documents. This could be just a simple code comment or

"Various MVC (Model-View Controller) framework-based components may be easier to test with other testing frameworks, some of which extend JUnit"

a complex mechanism where each test annotates references to the use cases and features that it tests, and a test failure causes a notification to be sent to the responsible developer. The important thing is to have the reference in the code and to maintain the traceability.

Developing Runtime Event Diagrams

To understand what the test covers, you need to understand how the code being tested functions, and how the various static classes work together to form the dynamic object graph that defines the program's state during the test.

There are many ways to model such behavior, including Granovetter Diagrams and Object Interaction Diagrams. The basic idea is to pictographically explore the code to understand what runtime parts are involved during the test. These techniques can all be described under the moniker Runtime Event Diagrams because they show what events occur while the program is running, rather than what events the classes could theoretically handle. These diagrams are important for a number of reasons.

First, these diagrams expose the code in a way that can easily be understood at a high level, and consequently provide useful documentation. This documentation is different than the documentation inlined into the code. These diagrams show the runtime behavior of the code, where the operation of the code's actual functions takes place, and where the system is more easily understood; most design patterns and other architectural decisions are much more easily understood in terms of objects and references than classes and fields.

In addition, these diagrams catalog what parts of the code are being exercised by any given test, and determine if that test will be affected by future changes to arbitrary code. When the developer is certain that test A utilizes subsystems B, C, and D, she is also certain that changes to B, C, or D will necessitate testing A again (to ensure backwards compatibility).

A good tactic is to attempt to model the system in as few steps as possible. In general, the actual calls being made are irrelevant; the important aspect is how the system works together to achieve the desired goal. This goal can be achieved by using a simplified modeling system that shows only general interactions between objects, with natural language descriptions of what is occurring in the various interactions.

After a Runtime Event Diagram is drawn, it can be incorporated into the class documentation. Note that a few limitations of the diagrams make them more resilient to class modifications. First, the method names are generally not used because they may change over time. Instead of method names, the more general and more easily understood natural language descriptions are used. Second, the diagrams are mainly about the interaction between the parts of the system. This is a high-level architectural design decision, and is less likely to change over time. Lastly, the diagram is built in terms of types, not specific classes. As long as the general types remain the same – which they are likely to do to retain the protocols used to interact with them – the graphs need not be updated.

Once the diagrams are created, they can be used in a variety of ways. For instance, a diagram can be used to gain a quick overview of how a system works and how it uses its interconnected parts to achieve its goals. This is a sort of simplified UML which describes only the system parts that really matter at a glance: instances, their types, the other instances which they reference, and the work being performed by the group.

These diagrams can also be used to gain insight into the system's complexity and how it could be simplified. To identify ways to simplify the system, look for objects that are used in the system once or twice, and investigate where they might fit better. Also, look for repetitive tasks, and try to encapsulate them into a method or a class.

However, the diagrams' most important use is to facilitate testing. By encapsulating the system's state, the diagrams can help solve problems that are occurring with the system. The diagrams' information about what should be happening can be referenced later when a problem appears. In this case, it is simpler to identify what went wrong because it is simple to compare the program's current state with its expected state. Fixes within small components should not change the overall architecture, and by using the Runtime Event Diagram that already exists, you can ensure that the system does indeed still function as expected. Moreover, by showing the system as it exists and works at present, the Runtime Event Diagram will demonstrate how the system can be modified when a more important component must change. By defining the system's behavior as a whole and how it is expected to function, the Runtime Event Diagram becomes a sort of architectural unit test. When the system changes, the changes can more easily be vetted to ensure that the proper functionality is maintained.

The diagrams should be used in situations when the details often obscure the bigger picture. Their high-level nature can be leveraged as insight into what design patterns are being used, or what AntiPatterns are showing themselves. Many other uses are possible, and when a Runtime Event Diagram, test case specification and use case specification fails to capture the necessary detail, it provides a road map that can be used to jump directly into the code.

Leveraging Functional Tests for Regression Testing

Finally, to boost your ROI on your functional testing efforts, configure a process to run these tests automatically in concert with your automated build process. Such a process not only functionally tests code, but also performs regular regression testing at the same time. Most modern development projects involve building upon a large existing code base. If the code base lacks sufficient tests, the team has no practical way to determine if the modifications broke any existing functionality. As a result, such code is difficult to extend or optimize. In contrast, developers with a regression test suite of comprehensive functional tests can improve and extend code without fear of introducing problems that could go undetected. After all, there is nothing like the pleasure of being able to run a regression suite and know that everything is still working as expected.

JDBC 4.0

XML, performance, and more

by John Goodson and Mark Biamonte

It's been over three years since the JDBC Expert Group held its first meeting to gather requirements, requests, and pipe dreams for the JDBC 4.0 specification. In that meeting, we discussed a wide variety of topics, including performance enhancements, clarifications on the existing JDBC 3.0 specification, and Ease of Development features. Unbelievably, everything but the kitchen sink ended up making it into the release. In this article, we'll look at several key features that made the enhancement list for JDBC 4.0, and we'll discuss why those features are important.

At the time of this publication, the JDBC 4.0 specification should be close to shipping as part of Java SE 6.0. The key goals of the JDBC Expert Group were to align with the most important features of the SQL 2003 specification, to provide constructs that improve developer productivity (sometimes called Ease of Development or EOD features), to fine tune pooling constructs, and to improve scalability. While many aspects of the JDBC 3.0 specification were somewhat limited, the new additions to the JDBC 4.0 specification apply to a wider audience. Additionally, the most common complaint about the JDBC specification is that it did not provide enough specifics. This vagueness often led to different implementations between JDBC driver vendors. In addition to the wealth of new features, JDBC 4.0 includes hundreds of fixes for bugs that have been addressed and clarifications stated to eliminate ambiguity among implementations.

XML Support

One of the most useful new features in JDBC 4.0 is support for the SQL 2003 XML data type. Support for the XML data type is also the feature that has changed the most since the initial public draft of the JDBC 4.0 specification. This draft introduced support for XML data types in the database, Java XML bindings, and SQL/XML extensions to the SQL grammar. Based on feedback from both the JDBC and XML communities, the JDBC interfaces for working with XML data have been enhanced and expanded significantly.

Today, applications must use either JDBC driver extensions or the Clob and Blob interfaces to transfer XML data to or from the database. Using JDBC driver extensions makes it hard to write a portable database application. Using the Clob and Blob interfaces limits the application to working with string representations of XML data and, in many cases, requires vendor-specific extensions to the SQL grammar to tell the database whether to return the data as a character or binary representation of the XML string data.

Now, the new JDBC data type, SQLXML, is part of the JDBC 4.0 specification. Applications can use the getTypeInfo() method to determine if their database supports a native XML data type. For example, using getTypeInfo() against a Microsoft SQL Server 2000 instance does not return a result row corresponding to the SQLXML data type, indicating that there is no native XML data type available for that particular database. In contrast, using getTypeInfo() against a Microsoft SQL Server 2005 instance returns a result row, indicating that an XML data type is available. Additionally, it returns information indicating that the native type name is XML. From this information, applications can create tables that contain columns of the XML data type.

To allow applications to populate data into XML columns and retrieve data from those columns, JDBC has been expanded to include native Java bindings for XML. In the initial public draft, the JDBC Expert Group defined bindings for Java strings and StAX. The thinking at the time was that this subset of XML representations could be used to easily construct other XML representations. However, feedback from the community made it clear that there are many applications today that rely on DOM and SAX and that any JDBC XML solution must provide support for those and other XML representations. The expert group went back and totally reworked the SQLXML interface definition.

The definition of the SQLXML interface in the proposed final draft of the JDBC 4.0 specification now includes support for generating and retrieving a character or binary representation of XML data as a Java String, a character stream, or a binary stream. More importantly, the SQLXML interface includes methods for working with the Source and Result interfaces defined in the javax.xml.transform package. These methods provide flexibility for supporting any XML representation for which there is a javax.xml.transform Source or Result implementation. Additionally, supporting the Source and Result interfaces allows a JDBC driver to easily become an end point of XSLT transforms or XPath evaluations.

John Goodson, Vice President of Product Operations, leads the product strategy, direction, and development efforts for DataDirect Technologies. For over ten years, John has worked closely with Sun and Microsoft on the development and evolution of database connectivity standards including J2EE, JDBC, .NET, ODBC, and ADO. His active memberships in various standards committees, including the JDBC Expert Group, have helped John's team develop the most technically advanced data connectivity technologies. John holds a bachelor of science in computer science from Virginia Tech.

john.goodson@datadirect.com

At a minimum, the JDBC 4.0 specification requires JDBC drivers to support the Source and Result interfaces shown in Table 1.

To create a Java construct that can be used to process XML data, an application can create a SQLXML object using the Connection.createSQLXML() method. The object that is created does not contain any data initially. Data is added to the object using one of the following methods:

- Calling setString()
- Using the Writer returned from setCharacterSteam()
- Using the OutputStream returned from setBinaryStream()
- Associating a Result with the SQLXML object using setResult()

Listings 1 and 2 illustrate how an application can use these techniques to insert a row containing XML data. The examples used in this section work on a table that has a column named id of type INTEGER and a column named xmlCol of type XML.

Similarly, applications can retrieve XML data from a SQLXML object using the getString( ), getCharacterSteam( ), getBinaryStream( ), and getSource( ) methods. Listing 3 illustrates how an application can query a column of the SQLXML data type, create a SQLXML Java binding using the getSQLXML() method on the result set, and retrieve a StAXSource object that is used to process the XML data.

SQL 2003 also includes extensions to the SELECT syntax that allow you to construct XML results from tabular columns. Listing 4 shows a simple example of how to create a SELECT statement that produces a result set containing two columns: a CustId column of type integer and a CustInfo column of type SQLXML.

The SELECT statement uses the new SQL/XML extension XMLELEMENT to process multiple base columns into a single XML result column. JDBC 4.0 also has been expanded to support using database metadata methods to determine which SQL/XML constructs are supported on the connection. Applications can execute any supported SELECT statement with SQL/XML extensions to produce SQLXML result columns that can be processed using the new XML Java bindings.

Connection and Statement Pooling Enhancements

Today, if you have a JDBC application deployed inside an application server or Web server, there's a good chance it uses connection pooling, statement pooling, or a combination of both to obtain better application performance. Pooling is great — except it's not tunable, it's hard to map end users back to connections in the pool, and if a connection ever becomes invalid inside the pool, removing only that connection from the pool is nearly impossible. JDBC 4.0 addresses all these drawbacks.

The architecture of many Java application servers or Web servers dictates that database interaction occurs as a result of an incoming message, a user clicking a button in a Web browser, or through some other real-time event. In each of these occurrences, a database connection is usually established, one or more SQL statements are executed, results are processed, and the connection is closed. In this type of architecture, the response time for the application is limited by the response time of the connection attempt. That is, connecting to a database is one of the most performance-expensive operations that a JDBC application can do. A connection involves multiple network round-trips between a JDBC driver and a database server to perform the following actions:

- Establish memory on behalf of the database user in the database server
- Authenticate the user
- Negotiate details between the JDBC driver and the database server, such as code page settings, getting the database version, and determining the optimal database protocol packet size

To limit this overhead, most Java environments use a JDBC connection pool. A connection pool manager establishes multiple database connections at system startup and keeps these connections available. When an application needs to establish a connection, the pool manager assigns one of the pre-allocated connections to the application instead of going through the time-consuming process of establishing a new physical connection to the database. When the application is finished with the connection, the connection is returned to the pool manager's cache of connections. In this type of environment, the JDBC driver and the database are not aware that the application connected and disconnected. When a connection is established, the pool manager loans out a connection and when a connection is closed, the loaned connection is returned to the pool. To the JDBC driver and to the database, the connection has been active since the system was started.

Connection pooling works great until there is a problem you need to investigate. When the response time of your database queries takes minutes instead of milliseconds, your application server suddenly starts to run out of memory or CPU cycles and your database appears to be "hung." Another possibility is that when you try to monitor the status of your applications, you find that "some JDBC connection" is using all the CPU and that the facilities available to help you find the culprit are not very good. Once a JDBC connection is established, the tracking mechanism between the physical connection and the application's use of the logical connection is lost. The connection pool manager assigns physical connections in the pool to any application that meets authenti-

Mark Biamonte is Program Manager for DataDirect's Connect for JDBC product. He is responsible for defining the technical features and future direction of the Connect for JDBC product. Mark has over 20 years of experience designing computer hardware and software. He has been working with Java and JDBC for over 5 years and database APIs for over 7 years. He is currently an active member of the JDBC Expert Group defining the next version of the JDBC specification. Mark holds a master of science in electrical engineering from Worcester Polytechnic Institute in Worcester, Mass and a bachelor of science in electrical engineering from the University of Vermont in Burlington, Vt.

"To allow applications to populate data into XML columns and retrieve data from those columns, JDBC has been expanded to include native Java bindings for XML"

Source Interfaces	Result Interfaces
javax.xml.transform.dom.DOMSource	javax.xml.transform.dom.DOMResult
javax.xml.transform.sax.SAXSource	javax.xml.transform.sax.SAXResult
javax.xml.transform.stax.StAXSource	javax.xml.transform.stax.StAXResult
javax.xml.transform.stream.StreamSource	javax.xml.transform.stream.StreamResult

Table 1 JDBC 4.0 required Source and Result types

cation requirements; the pool manager does not keep any statistics on the application requesting a connection, and the connection itself is a black box to the application. In other words, if you are using a monitoring tool and see that a JDBC connection is "bogging down the system," it's not possible to track down which JDBC application is actually invoking the driver.

As we stated earlier, connection pooling is usually provided by the application/Web server, so a connection request from an application is not sent to a driver, but is instead sent to the pooling component inside the server. For a JDBC driver to associate an application to a connection, it must be involved in the connection establishment process, which does not happen when using JDBC 3.0-compliant connection pool managers.

JDBC 4.0 has added the setClientInfo() and getClientInfo() methods to the connection interface to solve many of the problems mentioned above. After connecting, an application can call setClientInfo() to associate client-specific information to the JDBC connection object, such as application name, site name, and department name for the JDBC connection. The setClientInfo() request is executed in the JDBC driver and not in the connection pool manager. The JDBC driver then passes along this information to the database server. In this way, monitoring tools can retrieve client information for specific database connections from either the JDBC driver or from the database server to help pinpoint where problems are occurring.

Another problem we often see in today's popular JDBC connection pool implementations is that there is no good way for a connection pool manager to determine if a database connection has become unusable. Typically, if a pool manager detects that any single connection in the pool has become invalid, the pool manager terminates all connections in the pool regardless of whether they're usable. After the pool is flushed, the pool manager re-initiates the pool with new connections. Flushing the pool is a drastic process that results in the potential loss of business logic, poor performance, and, typically, irate users. Some pool managers erroneously use the Connection.isClosed() method to check the state of a database connection, but the intention of isClosed() is to check if a connection is open or closed, not to determine if the connection is still usable. A new method has been added, Connection.isValid(), to allow pool managers to specifically request from the driver if a connection is still usable. If a connection is invalid, the pool manager can discard only the marred connection rather than the contents of the entire pool.

In addition to the connection pooling mechanism previously discussed, JDBC 3.0 introduced a statement pooling mechanism to cache prepared statements. The statement pool manager, which can be part of the JDBC driver or part of the application/Web server, keeps prepared SQL queries in a cache that can be reused by applications. In the same way that a connection pool manager keeps performance-expensive database connections in a pool for "loan," the statement pool manager keeps SQL prepared queries in a cache that can be loaned out when an application attempts to use a SQL query that matches one in the pool.

JDBC statement pooling provides performance gains for JDBC applications that execute the same SQL statements multiple times in the life of the application. Most applications have a certain set of SQL statements that are executed multiple times and a few SQL statements that are executed only once or twice during the life of the application. Unfortunately, existing JDBC statement pooling implementations give no weight to a SQL statement that's executed 100 times versus one that's executed only twice. Either a statement goes into the pool, potentially displacing another statement from the pool, or there is no pool. JDBC 4.0 provides a more granular level of statement pooling by allowing applications to provide directives to the pool manager about whether a SQL statement should be pooled.

The PreparedStatement interface has been expanded by the addition of two new methods: isPoolable() and setPoolable(). The isPoolable() method returns a Boolean flag that denotes whether the SQL statement identified on the PreparedStatement object should be pooled (by default, a statement is poolable when it's created). Applications specifically can request that a statement not be pooled by calling setPoolable(false). Using these constructs, application architects gain more control over the performance aspects of their JDBC applications. Queries that are reused are pooled and provide optimal performance, and queries that are used infrequently do not affect the pool.

National Character Set Support

When the JDBC 1.0 specification was developed, the primary goal was to make the specification fit the Java model of programming and make it easy to use. When the topic of national language sets was discussed, it was decided to postpone introducing these types into the specification because they were complex to explain, not well understood,

"JDBC statement pooling provides performance gains for JDBC applications that execute the same SQL statements multiple times in the life of the application."

and the hope was that JDBC drivers could mask most differences, for example, those between NCHAR and CHAR. Besides, Java was Unicode anyway.

It turns out that most JDBC drivers do need to know when sending character data to a database if the type the database server is expecting is a Unicode type (or National Character). It also turns out that most JDBC drivers can't figure out what the database is expecting without expensive network round-trips to the database server. JDBC provides only one type of binding using setString(), setCharacterStream(), and setClob(). If a SQL parameter corresponds to an NCHAR type, the application binds the parameter using setString(). Similarly, if the parameter corresponds to a CHAR type, the application uses setString() also.

To compensate for this deficiency, JDBC drivers typically adopted one of the following three strategies:

- Always send the character data to the database server in "safe" mode, which typically results in a performance penalty when the data doesn't match the format expected on the database server
- Provide connection properties that must be set to correspond to the types being used on the server – clearly a problem when both types are used
- Assume all data is ANSI and can possibly fail with data corruption if the database types are Unicode (or National types)

None of these options benefit application designers, so the JDBC 4.0 specification now provides mechanisms to denote National Type characters – (setNString(), setNCharacterStream(), setNClob(), and setObject()); and ANSI characters – (setString(), setCharacterStream(), setClob(), and setObject()).

Other New Features

We've touched on only three new features of the JDBC 4.0 specification, but the specification contains over 20 new features as well as hundreds of valuable specification clarifications. The JDBC 4.0 specification also includes support for extended SQLException hierarchies, a new ROWID data type, improved management of Clob and Blob objects, an improved mechanism for installing and recognizing JDBC drivers on a system, and more.

One of the innovations removed late in the specification process were the Ease of Development features, including annotation support. Look for annotation support soon after the release of Java SE 6.0. Take a good look at the JDBC 4.0 specification at http://www.jcp.org/en/jsr/detail?id=221 for details about the features we've mentioned in this article as well as all the features we didn't have space to include. As you start to use JDBC 4.0, remember that some of the new features are targeted for JDBC pooling components usually available in an application server or Web server while others are targeted for JDBC drivers. Check out the components you're using to make sure that they support the parts of the specification you're interested in and start reaping the benefits of JDBC 4.0.

Listing 1: Inserting Data with a SQLXML Object – XML Data Set Using setCharacterStream()

```
sql = "insert into xmlTable values (?, ?)";
PreparedStatement prepStmt = con.prepareStatement(sql);

String xmlStr = "<MyXMLData> ... </MyXMLData>";

SQLXML sqlXML = con.createSQLXML();
Writer xmlWriter = sqlXML.setCharacterStream();

xmlWriter.write(xmlStr);
xmlWriter.close();

prepStmt.setInt(1, 7);
prepStmt.setSQLXML(2, sqlXML);
prepStmt.executeUpdate();
```

Listing 2: Inserting Data with a SQLXML Object – XML Data Set Using DOMResult

```
File xmlFile = new File("Addresses.xml");

// Get a DOM Document
DocumentBuilderFactory docBuilderFactory =
   DocumentBuilderFactory.newInstance();
docBuilderFactory.setNamespaceAware(true);

DocumentBuilder docBuilder = docBuilderFactory.newDocumentBuilder();
Document doc = docBuilder.parse(xmlFile);

sql = "insert into xmlTable values (?, ?)";
PreparedStatement prepStmt = con.prepareStatement(sql);

sqlXML = con.createSQLXML();

DOMResult domResult = (DOMResult) sqlXML.setResult(DOMResult.class);
domResult.setNode(doc);

prepStmt.setInt(1, 9);
prepStmt.setSQLXML(2, sqlXML);
prepStmt.executeUpdate();
```

Listing 3: Retrieving Data with a SQLXML Object – XML Data Read Using an XMLStreamReader

```
String sql = "select id, xmlCol from xmlTable";
resultSet = stmt.executeQuery(sql);
while (resultSet.next()) {

   id = resultSet.getInt(1);

   SQLXML sqlXML = resultSet.getSQLXML(2);
   StAXSource staxSource =
      (StAXSource) sqlXML.getSource(StAXSource.class);
   XMLStreamReader xmlReader = staxSource.getXMLStreamReader();

   // Process the StAX events using xmlReader
}
```

Listing 4: Constructing an XML Result

```
Select
   c.CustId,
      xmlelement(name customer,
         xmlelement(name name, c.Name),
         xmlelement(name address, c.Address)) as CustInfo
from Customers c
```

CustId	CustInfo
1	<customer> <name>Woodworks</name> <address>Baltimore</address> </customer>
2	<customer> <name>Software Solutions</name> <address>Boston</address> </customer>
...	...

Patterns In Action

Pattern-driven software engineering

by Jochen Krebs

Object-oriented software engineering (OOSE) without design patterns is like cooking without a recipe. Patterns guide us with ingredients and step-by-step instructions for assembling the solution to a recurring problem. In the same way we rely on recipes in cooking, we experience patterns as repeatable, proven solutions, and software engineering becomes more reliable and successful.

As in the culinary arts, where chopping and cutting techniques are prerequisites for mixing and flavoring dishes, there are many design patterns for all sort of challenges - basic, intermediate, and advanced - depending on your needs. However, food recipes often contain references to other recipes that go well with the main dish, thus enhancing the entire meal.

This article will focus on exactly these pattern relationships, combinations, and variations. It's all part of an emerging trend we might call "pattern-driven software engineering." The examples I provide are visualized in UML and would eventually be transformed into code (e.g., Java). Because patterns do not only affect the structure and dynamics of classes and objects, this article will conclude investigating the role of patterns in a service-oriented architecture (SOA).

The Concept of Patterns

Patterns emerge as software engineers begin to notice recurring problems. If you design software and you face a situation in which you ask yourself, "Gee, I can't be the first person facing this problem!" - your search for a pattern has just begun. Once you find and apply a pattern, your solution will not only benefit from the knowledge gained in the past, but this pattern might also open a door to related patterns. An individual pattern works in its described context and offers a variety of related patterns that can improve the quality of your solution even more. Eventually, one design could be a starting point for an entire pattern-driven design process.

Before we discuss the relationships among patterns, let's explore that culinary metaphor a bit and take a look at some individual patterns.

I'll describe a typical TV cooking show to help explain software patterns and their relationships. The goal of the show is to demonstrate the preparation of a specific meal. On most cooking shows, however, we find cups and bowls in front of the chef, with ingredients such as onions already prepared. That's because the expert cook doesn't need to illustrate the chopping of onions in front of the TV audience; it would be boring. Prior to the taping of the show, the chef has probably asked his subordinates for some quantity of "finely chopped onions," the same ingredients used in many recipes. What's important here is that the chef doesn't need to communicate the actual cutting technique; he simply asks for the well-known result, a standard cup of chopped onions.

Software engineers make use of such basic patterns, too. Some of these patterns, such as the General Responsibility Assignment Software Patterns (GRASP) (*Applying ULM and Patterns* by Craig Larman), are so fundamental that many other patterns make use of them. Basic design patterns organize and control communication or creation, or they establish visibilities among objects. Basically, in an object-oriented system, objects communicate with each other through messages. Therefore all these messages (a.k.a. responsibilities) need to be assigned by the software engineer to build a flexible and maintainable system. Based on that fact, object-oriented software engineers constantly ask themselves the same basic question: "Who should talk to whom?".

The Problem Scenario

For the remainder of this article, I will illustrate various approaches to pattern usage through the scenario of a change request to a timesheet application, where the change has to do with the timesheet approval process. Figure 1 shows a typical situation for an object-oriented designer, where a specific business rule requires identifying whether the timesheet is approved or not. The question ("Are you approved?") and the answer ("yes" or "no") are determined, but the questions remain: who should receive and who should send the message?

Even for very basic design situations like the one described in Figure 1, we can make use of fundamental design patterns; for example, asking the GRASP patterns for help.

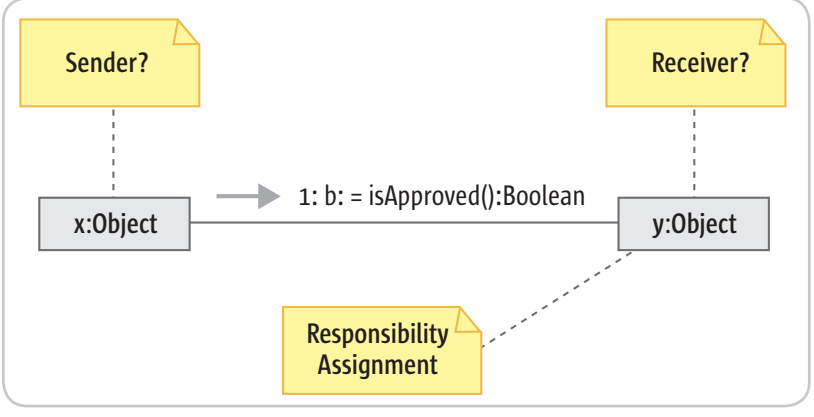


Figure 1 Responsibility assignment

Jochen (Joe) Krebs, http://www.jochenkrebs.com, is a method engineer within the Rational Brand for IBM. He develops content for the Rational Unified Process, OpenUP, and other agile software engineering processes. Prior to his current role he was responsible for successful enablement of Rational products and services for clients in the financial sector. Before joining IBM Rational he worked as an instructor and senior consultant with a focus on project management, requirements management, software engineering processes, and object-oriented technologies using Smalltalk and Java. He holds his MSc in computing for commerce and industry at the Open University.

jochen.krebs@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

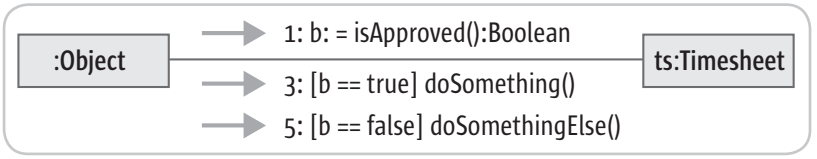


Figure 2 A UML example of violating the Expert and Polymorphism patterns

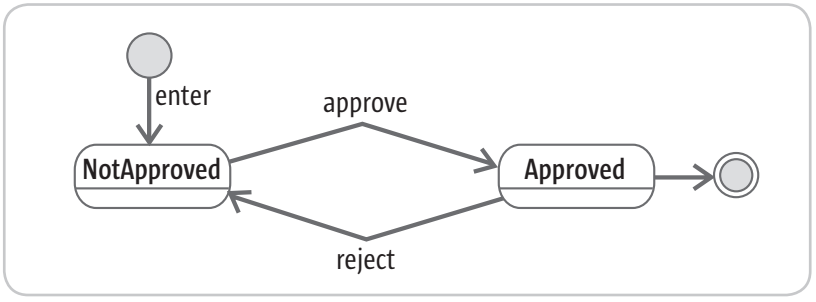


Figure 3 UML state-machine diagram for timesheet (two states)

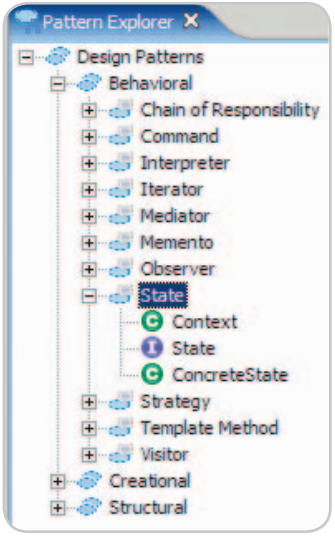


Figure 4 State pattern and participating classes within the RSA pattern explorer

In the TV cooking show, the chef is using a fundamental pattern - chopped onions - to assemble a more complex pattern of his own, the meal itself. The level of the pattern has been elevated from a single set of techniques to a dish that comprises other fundamental techniques. The recipe has a name; for example, tomato sauce. It is the chef's responsibility to decide how many onions he uses and how he prepares them. The problem now moves to a higher level, from chopping onions to making a good tomato sauce. The chef begins applying his own pattern, the recipe, which contains other patterns (for sautéing, chopping parsley, etc.). The experienced chef applies a pattern, in a sense, as a way to present food nicely, focusing on color, texture, and style.

Software design patterns are not different. In addition to the fundamental GRASP patterns, engineers make use of more elevated patterns, such as Gang of Four (GoF) (*Design Patterns* by Gamma, Johnson, Helm and Vlissides) or architectural patterns. Now that most software engineers graduating from universities are grounded in OO principles, the software development industry has begun to raise the level of pattern adoption from the level of problem-solving techniques to problem-prevention techniques. I will use the *Design Patterns -Reusable Objects* (from the Gang of Four) as a design pattern catalog to demonstrate the pattern relationships and use the IBM Rational Software Architect (RSA) pattern catalog to illustrate the examples.

Let's get back to our initial scenario illustrated in Figure 1, in which we plan to build a timesheet application with a focus on an approval process. The designer needs to identify whether a timesheet is approved or not. In this case, it seems almost enough to simply add an attribute called *is Approved* to the *Timesheet* object, which contains one of the boolean values, *true* or *false*. The problem with this solution is, however, that the attributes of the object can change, and depending on the content of the attribute we would need to determine the type of message that will be fired. If we want to add another option - for example, *Submitted* - the boolean attribute, which allows two possible values *true* or *false*, does not accommodate this design approach anymore. With the introduction of the *Submitted* state the original design (built for two values) would break and the entire business logic would require us to reevaluate our initial design.

Later, I will demonstrate how smooth the transition can be from a two-states design to a three-states design, when patterns are applied. As illustrated in Figure 2, our new design approach would violate two fundamental design patterns, Expert and Polymorphism (according to Larman, op. cit.) and would unnecessarily couple one object with the business logic that belongs to another object.

The boolean value approach would not only violate fundamental design patterns, it would also increase the maintenance burden for software engineers because the design for the *Timesheet* object could easily break and the entire object would need to be retested with every change.

Translating the UML design from the code below would generate a Java structure like this example, violating Expert and Polymorphism.

```
....
  if (b == true)
  {
          ts.doSomething();
  }
  if (b == false)
  {
          ts.doSomethingElse();
  }
  ....
```

One Solution: The State Pattern

The GoF pattern catalog offers a possible solution for our design challenge. The pattern is called *State*.

First, we verify that the pattern meets our needs; then, we read the intend, application, and consequences sections of the pattern. Because the pattern says that it "Allows an object to alter its behavior when its internal state changes. The object will appear to change its class [GoF]," we go ahead and apply this pattern to our problem.

One of the benefits of applying the *State* pattern is that it can resolve the if-statement situation difficulty shown in Figure 3 by isolating the various states. The UML state-machine notation helps us depict and investigate the various

states. Initially our timesheet was fairly simple and we isolated two states out of our existing structure, *Approved and Not Approved*, as shown in Figure 3.

Instead of asking the object which value is nested in an attribute (in our case *is Approved*) and make a decision based on that (which violate the principle of polymorphism) we instead tell the object what to do and simply send the message to it and let the *Timesheet* object deal with the event. What we would like to design is some way to send a message, as shown below, where *ts* is a *Timesheet* object. This is a new responsibility assignment for timesheet (Java)

```
....
ts.approve();
....
```

After we isolate the various states, remove the if-construct from the *Timesheet* object, and assign the three responsibilities (enter, approve, and reject), we then want to apply the *State* pattern to our solution. Using the RSA pattern explorer we navigate to the *State* pattern (See Figure 4), which shows us the participating classes in the pattern.

In order to get an overview of the structure of the *State* pattern, the pattern explorer provides us the layout shown in Figure 5.

The cookie-cutter solution for the *State* pattern needs to be adjusted to accommodate our application's specific needs. After dragging the pattern from the pattern explorer directly into our workspace, we can assign the participating classes from our application-specific class model. Figure 6 shows the *Timesheet* as a context object, the Java interface *ITimesheetState* for the *State* and both concrete states from our timesheet application (*Approved* and *Not Approved*).

The dynamics of this pattern are shown in the code below, using Java. After the message *approve()* has been sent to the Timesheet object, it takes the message and delegates it to its state and provides a pointer back to itself (the *this*-parameter). Below is the message delegation from the context to the State Object.

```
....
state.approve(this);
....
```

After the message *approve(this)* has been sent, the state which at runtime is located in the *State* object will handle the event (which is truly polymorphic). For example, the *Not Approved* state would implement the *approve(ITimesheetState state)* message. Below is the concrete state method implementation – Not Approved.

```
public void approve(Timesheet ts)
{
      ITimesheetState newState = new
        ApprovedState();
      ts.setState(newState);
}|
```

To support the polymorphic approach, we need to assign the approve responsibility also to the *Approved* state, as shown below, even though we will not do anything in this particular situation. The code below shows the Concrete state method implementation – Approved.

```
public void approve(Timesheet ts)
{
    // do nothing|
}
```

Now that the *State* pattern[GoF] has been applied, let's see what happens in our one-pattern design if a requirements change occurs: for example, stakeholders need to be able to submit their timesheet after the time has been entered, and request approval. The following state machine diagram in Figure 7 shows two new states, *Entered and Submitted*, which replaced the previous state *Not Approved* to accommodate this requirement change.

The UML Design Class diagram in Figure 8 depicts the changes caused by the new requirement to the class model. Even though new state classes and messages have been introduced and one state has been removed, the changes are still very manageable. The most important point to be made is that the *Timesheet* has not been changed at all. It still keeps passing all the messages it receives to its actual state. That is a tremendous improvement to our if-else construct from the Java example,violating Expert and Polymorphism, because the area of concern from a testing perspective has shifted away from the *Timesheet* object to its states.

Pattern-Driven Development

In the previous section, I illustrated a design problem, applied a common solution (the *State* pattern) to it, and pointed out the advantages of the design using the pattern (maintainability and flexibility). In a pattern-driven solution, a designer will not only apply a pattern when a problem occurs, but will drive

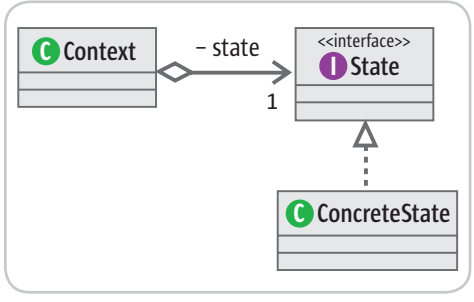


Figure 5 State pattern structure within the RSA pattern explorer

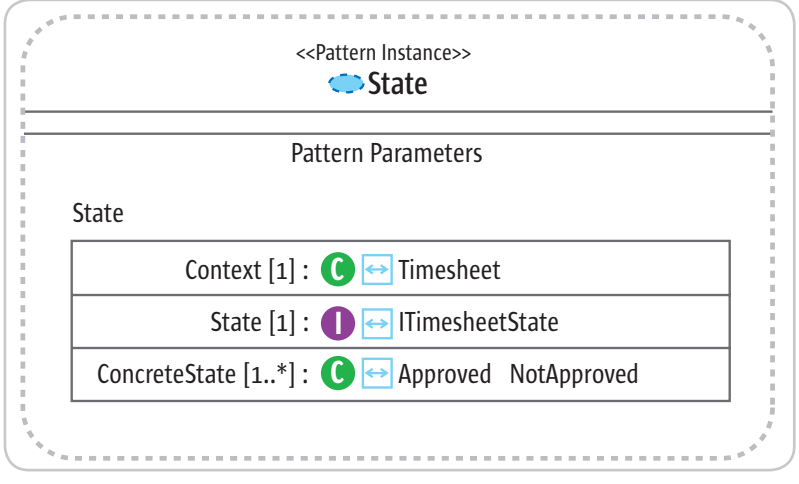


Figure 6 Applied state pattern in RSA

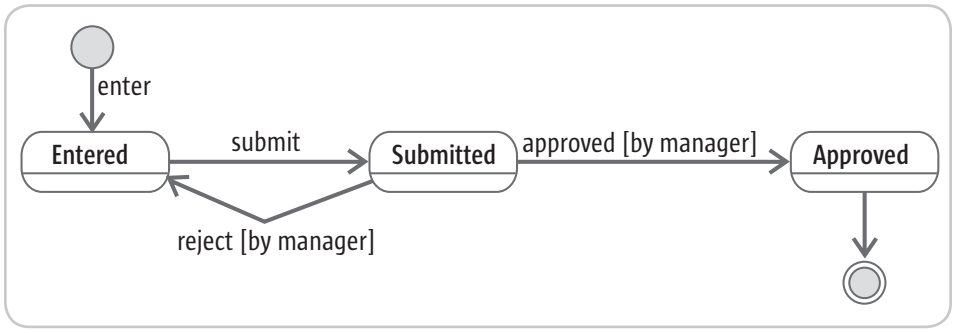


Figure 7 UML state machine for timesheet (more states)

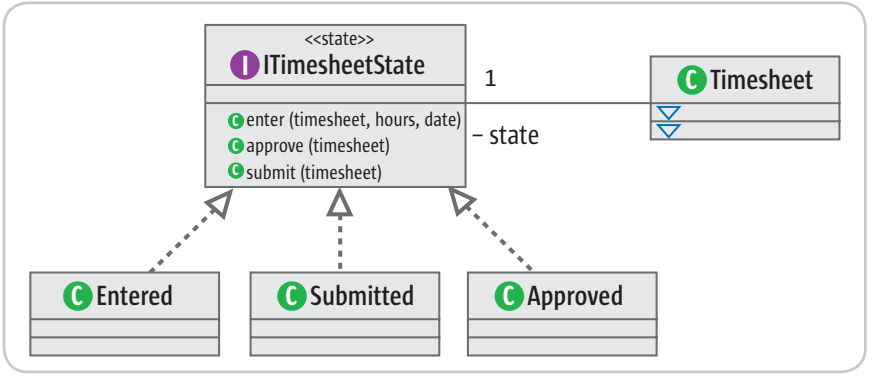


Figure 8 Partial UML design class diagram (timesheet and new states)

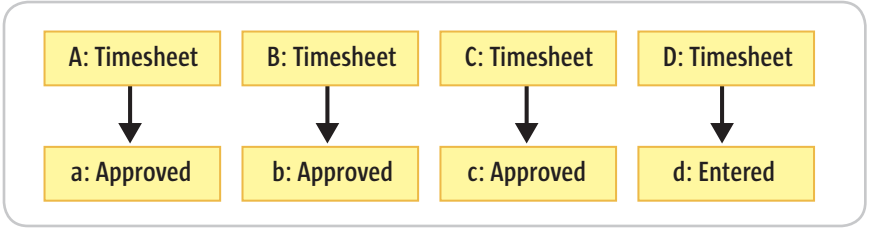


Figure 9 Object model prior to Flyweight pattern

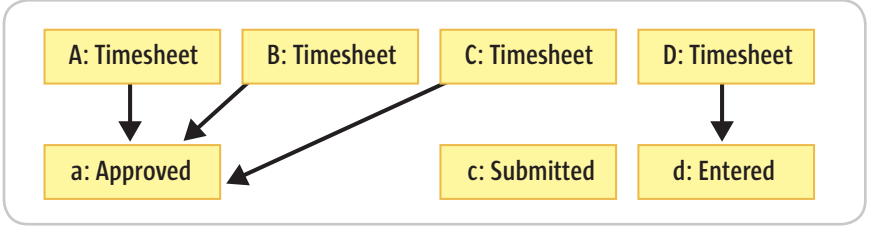


Figure 10 Object model after applying the Flyweight pattern

the entire design by using patterns. This approach is slightly different, because it assumes that the designer works actively with design pattern catalogs and uses the relationships between those patterns. The patterns within a catalog are usually grouped according to a chosen template. The GoF pattern template, for example, has *Name and Classification, Intent, Also Known As, Motivation, Applicability, Structure, Participants, Collaborations, Consequences, Implementation, Sample Code, Known Uses* and, last but not least, *Related Patterns*.

The *Related* patterns section within the pattern template contains important clues to other patterns that might be applicable in the context of the object. For example, according to the pattern catalog, the *State* pattern is often related to *Flyweight* and *Singleton[GoF]*. This information must trigger a new set of questions to the software engineer - for example, "Are there any parts of my solution that could also benefit from the use of the *Flyweight* or the *Singleton* pattern?" - and cause the engineer to examine the existing approach.

Our solution using the *State* pattern currently has one drawback. If our timesheet system will handle, say, 5,000 timesheets in the approved state, we would also carry 5,000 instances of the *Approved* state. Also, with every state change, we would create a new instance of a new state and the Java garbage collector would need to collect the old state objects. This might not be very critical for our timesheet application, but in other scenarios this could be very expensive in terms of resources. Figure 9 shows just a small number of timesheets and associated states that would multiply thousands of times in our application, as it is thus far designed.

In our timesheet example, the states "approved, submitted. and entered" are good candidates for *Flyweight* objects because there is no need to add any additional attributes to one of the states to distinguish these instances. With the *Flyweight* pattern, we are actually able to improve the *State* pattern solution even further, as shown in Figure 10.

Our timesheet application now contains only three different state instances at any given moment, which increases maintainability and performance.

However, applying the *Flyweight* [GoF] pattern raises a new challenge for our designer. Instead of creating a new instance of a particular state object or carrying the flyweight objects around as parameters, we would like to achieve visibility to this one instance of state in that particular situation. The *Singleton* [GoF] pattern serves exactly this purpose. We could either implement each state as a Singleton, or create a factory of states which creates and manages the states. The latter method would increase maintainability even more through separation of concerns.

In pattern-driven development, the destination may be the origin for new patterns. For example, the State pattern often harmonizes with the *Flyweight* and *Singleton* patterns. However, *Flyweight* and *Singleton* have further associations to even more patterns, and

so on. For simplicity, I have limited my illustration of these pattern relationships to the GoF catalog; the *Also Known As* section of the pattern template opens the door to other catalogs as well.

Pattern-Driven Development in Service-Oriented Architectures

In service-oriented architectures (SOA), there are services, providers, and consumers, just as there are responsibilities, receivers, and senders in object-oriented architectures. The difference is that the application-centric design approach, (e.g. the timesheet application) is elevated to the orchestration of services aligning the IT system with, for example, the business processes. So far, our design for the timesheet application has been driven by technology and a requirements change. To include the timesheet application in a SOA, services must be exposed so that human and non-human interfaces can be consumed.

In SOA design, the isolated view of the timesheet application would be replaced by a view of how the services of the timesheet application harmonize with other services from other applications. Depending on business requirements, the timesheet application could be seen in light of an organizational payroll process or in combination with a cost-breakdown-structure for product management. Exposing or modifying existing services or creating new services becomes a critical task for the enterprise and application architect. A more flexible and maintainable system built with patterns is therefore a critical element for a successful SOA.

Earlier in this article, I showed how patterns promote the application of additional related patterns, even across catalogs. We also saw that patterns exist in various forms, from helping with design decisions on an object level to patterns on an application level (i.e., assigning responsibilities vs. GoF patterns). The pattern-driven approach helps provide flexible and maintainable services in an SOA, and the SOA itself can drive and stimulate the pattern-driven approach. Business modeling patterns as well as architectural/network patterns motivated through an SOA can drive pattern-driven development top-down, whereas the pattern-driven application design prepares a successful SOA bottom-up.

Conclusion

Finding a matching pattern for a problem not only presents a solution to a problem, but also means in many cases the beginning of a new search and further evaluation of related patterns. This type of search/discovery/exploration activity should be familiar to you if you've ever used an Internet search engine to explore a topic you only vaguely understand. You often start with terms you're not sure about, but as you see more accepted terms and areas of knowledge unfold in your result set(s), you gain insight into the "patterns" of thinking and solutions that exist. Soon, you are able to enhance your own queries, eventually expanding your original lines of thought.

When properly documented and cataloged, patterns provide a common roadmap, encourage engineers to investigate the problem space, and, more importantly, allow us to apply a set of proven solutions rather than only one particular solution. I have used a basic enhancement request to illustrate the impact of change to a pattern-driven solution for a system in maintenance mode. Following an iterative-incremental process model, projects face very similar situations during construction, and we can easily map the benefits of pattern-driven development to incremental improvements in the project.

The Rational Software Architect (RSA) provides capabilities to support a pattern-driven engineering process by starting with common design patterns (for example GoF), or by creating its own pattern catalog. Publishing a pattern catalog with RSA and sharing the library of patterns on an enterprise level increases adoption rate which results in more reliable and flexible IT design. The ability to react to and adapt to organizational change is a fundamental strategy for a SOA.

References

- Gamma, E., Johnson, R., Helm, R., and Vlissides, J.M. (1995). *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley.
- Larman, Craig. (2004). *Applying UML and Patterns - An Introduction to Object-Oriented Analysis and Design and Iterative Development* 3rd Edition. Prentice Hall.
- Alexander, C. (1979). *A Timeless Way of Building*. Oxford University Press.
- Cooper, J.W. (2001). *Java Design Patterns -- A Tutorial*. Addison-Wesley.
- Alpert,S., Brown, K., and Woolf, B. (1998). *The Design Patterns - Smalltalk Companion*. Addison-Wesley.

Joe Winchester
Desktop Java Editor

The Two-Dimensional Legacy of GUIs

Ted Nelson, inventor of, among other things, hypertext, once lamented that software development today is at the same evolutionary stage film making was at 100 years ago. Back in the 1900s, when the technology of film production was in its earliest stages, the cameraman was the person in charge because he was the one who understood the technology and could make it function correctly. The audience's sheer fascination with the magic of films was enough to captivate and hold their attention while the silent and blurred subjects grinned and gawked directly into the lens. Much has changed in the last hundred years though, and movie directors are now the ones in charge of making a film. It is they who decide every camera angle, every pan and zoom, every focus switch, and every light level in order to create the finished product. Their goal is to captivate the viewer, hold their attention, and suspend disbelief for 90 minutes or more. Cinema is a branch of the arts whose end product is thing of aesthetic quality.

There are striking similarities between cinema and GUI software, the most obvious of which is the fact that both interact with their users through a screen. It was developers at the Xerox Palo Alto Research Center (PARC) in the 1970s who realized that the eye is the highest bandwidth connection to the brain, and to interact with the user in the most efficient and natural way required using a terminal as more than just a way of receiving and displaying formatted text output. As a result of this epiphany, the first WYSIWYG editor was created and the concept of the GUI was born. Having the user interact with the information via scroll bars, push buttons, pop-up menus, check boxes, and so forth were all conceived 30 years ago. Each time a new whiz bang PC operating system is released or a software package created that will "revolutionize the way we work," I am always disappointed to peel back the hype and find that most of the effort occurred under the hood, leaving the front end largely unchanged with just a token rearranging of controls. Yesterday's dinner is re-served with fresh salad dressing and a quick 30 seconds in the microwave to make it smell fresh.

The problem is a deep one and can be traced back to the original Xerox developers who, although they were undeniably creative geniuses who shaped most of modern computing, also created the legacy that now holds us back. Their task back then was to create applications that supported laser printers; they needed a way for the user to see what the finished output would be before wasting toner and paper. WYSIWYG is more aptly an implementation of the acronym WYSIWYP or "what you see is what you print." For example, the scrollbar was designed to allow the user to deal with having their available viewing area smaller than the size of the underlying page, not as a

way of navigating through large lists of back-end data. All of our computing metaphors – such as copy and paste, applications on a desktop, folders with files, trash cans, and so forth – come from a paper-based view of the world. Ted Nelson writes that "like the fish that is unaware of water, computer users are blind to the 2D tyranny of paper." The problem occurs because everything we do in GUI software is predicated on the fact that we're using the computer as though it were a page of information.

What needs to be done then to break out of the goldfish bowl we're all currently swimming around in? We need to create applications that push the boundaries of software toward cinema, where the user experience is all that matters and our job isn't to be latter-day cameramen just arranging precanned widgets and controls designed to help some laser printer engineers.

Part of the problem lies in our psyche, who we are, and where we came from. In *Hackers and Painters* by Paul Graham (http://www.paulgraham.com/hp.html), he draws a parallel between the numerous people involved in creating a piece of art. One is the painter who has the inspiration, the talent, and the creative ability to produce the finished piece. To do this he has the knowledge of how to apply paints in the right way, the right combination, and the right technique to produce a work of art. The other person is the engineer at the paint factory. He has the knowledge of how chemicals can be mixed together to produce different products; paints that dry on the canvas and not in the tin; different colors, mediums, and so forth. The two disciplines are clearly different and neither the artist nor the engineer would be able to apply each other's trade. When the paint factory hires new recruits, they approach the chemical engineering department for graduates, and when the art gallery needs new material, they visit the art department. Software should be the same, where for a GUI application the people required understand how to communicate with users through a computer screen and how to convey information. These people are artists, not engineers. They are people who understand how to direct a movie, not those whose only skill is loading film in a camera and switching it on and off.

When this is understood it can be used to good effect, for example in the computer gaming industry where the production process has more in common with a movie studio than a traditional software house. Likewise, the plethora of special effects that occur in films, not to mention the raw technology underpinning CGI cartoon movies, makes the boundary between the two disciplines even more blurred. Computer games companies hire, and put in charge of production, creative talent and artists, while the engineer's job is relegated to cutting the code, pointing the camera, and making the canvas come to life in the image of the design team's ideas and vision.

What we need to do for desktop software is to learn from the games programmers, to recognize that the screen can do more than rehash a 30-year old paper presentation metaphor to the users, and see if we can inject more art, and less science, into the applications we produce.

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.

Real SOA

An overview of SCA and SDO

by Andrew Borley, Simon Laws, and Haleh Mahbod

A challenge facing many organizations is how to quickly and effectively react to frequent changes in business requirements, whilst improving productivity and reducing costs. To achieve this, you need a flexible infrastructure that can meet the demands of a changing marketplace and seize emerging opportunities. To address this challenge, Service Oriented Architecture (SOA) promotes an architectural approach that replaces rigid proprietary systems with heterogeneous, "loosely-coupled" services. The Service Component Architecture (SCA), along with Service Data Objects (SDO), makes this architectural concept a reality and provides the programming model to build SOA solutions for agile businesses.

SCA is a powerful and simple business level programming model which extends and complements prior approaches for implementing services based solutions. SCA defines how services can be described, assembled, and deployed in a meta-data driven fashion, independent of an implementation language and a deployment platform. The approach is based on the idea that each business function consists of one or more components brought together into a composite application. These, in turn, are composed into a network of services that create specific business solutions.

This article describes some of the key values of SCA by modeling an SOA based solution for a fictitious company called MostMortgage. We shall assume a simple business process in which an applicant signs up for a loan and provides his or her identity information and loan requirements. MostMortgage evaluates the new applicant based on their credit approval and searches for an appropriate mortgage rate.

By using the SCA programing model, MostMortgage's developer can build a solution for this problem quickly and effectively, separating the business logic from technology concerns and enabling re-use of existing applications. In this case, there is already a well understood credit appliction that can be re-used (CreditCheck); and MostMortgage has a subscription to a Web service that searches for the best loan rates (FindRates).

The solution developer completes the following steps:

1) Define the business logic for LoanApproval and AccountVerification;
2) Define references for each component (this identifies what other services, if any, the component is dependent on);
3) Define the services provided by each component, if any;
4) Assemble the components and choose the binding to be used.

The MostMortgage solution (as shown in Figure 1) is then ready for deployment.

Components can be implemented in any language supported by an SCA runtime, including BPEL, Java, Ruby, and C++. Outside of any program logic, these components can be assembled or "wired" into a composition using any appropriate binding, such as WS-* or JMS.

Let's update the MostMortgage application by making one technology domain change and one business domain change.

First, we improve the security of the calls to the CreditCheck component, which happens to run in a remote data center. The MostMortgage company developer need not be concerned about these new infrastructure requirements. SCA separates infrastructure capabilities from business logic and allows the security require-

> "The Service Component Architecture (SCA), along with Service Data Objects (SDO) ...provides the programming model to build SOA solutions for agile businesses"

Andrew Borley is an IBMer enjoying life working on the Apache Tuscany project. He's helping to define the Service Component Architecture (SCA) specification and is a committer on Apache Tuscany, developing implementations of SCA and Service Data Objects.

borley@uk.ibm.com

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

ments to be defined as policies during assembly. The resulting flexibility enables IT infrastructure policies to change at anytime without requiring a re-code.

Secondly, it is decided to introduce a bespoke rate optimization layer in front of the FindRate Web service (see Figure 2). MostMortgage can add value to the white box FindRate service by combining a mortgage account with in-house financial products. Here, the developer is involved, but he is able to reuse his previous work directly in a controlled and modular way by simply extending the assembly to include the new RateOptimizer component.

It is important not to forget the complexity introduced by handling data in such a heterogeneous network of services. A technology called Service Data Objects (SDO) addresses this problem. SDO offers a format-neutral API that provides a uniform way to access data, regardless of how it is physically stored. By using SDO, the solution developer will not pollute a business application with code to handle diverse choices of data access, such as JDBC Result Sets, JCA records, DOM, JAXB, and EJB entities.

SDO supports a disconnected style of data access and can record a summary based on any changes made to data objects. SDO's ability to maintain a summary of the changes made allows data transfers to include only the portion of data that has changed, therefore improving environments where bandwidth is a constraint. The change summary information can be used to resolve data access conflicts and concurrency issues.

SDO supplies a powerful yet simple programming model for data with first class support for XML and the ability to automatically persist data via the use of a Data Access Service (DAS). A DAS allows the data to be stored or retrieved from a relational database or another repository, and helps to link the SDO models to enterprise data storage.

SCA and SDO provide technologies that simplify the development of SOA solutions. SCA and SDO technologies work well together and independently. More detailed information about SCA and SDO will be available in future articles.

The importance of these technologies has led many vendors who experienced customer pain points to collaborate and develop specifications for SCA and SDO, and crystallize best practices that have been utilized for the past few years. More information about the Open SOA collaboration and its many participating vendors can be found at www.osoa.org.

You can try out Java and C++ implementations of the SCA and SDO technologies by visiting the Apache Tuscany open source project at incubator.apache.org/tuscany . Tuscany provides a simple

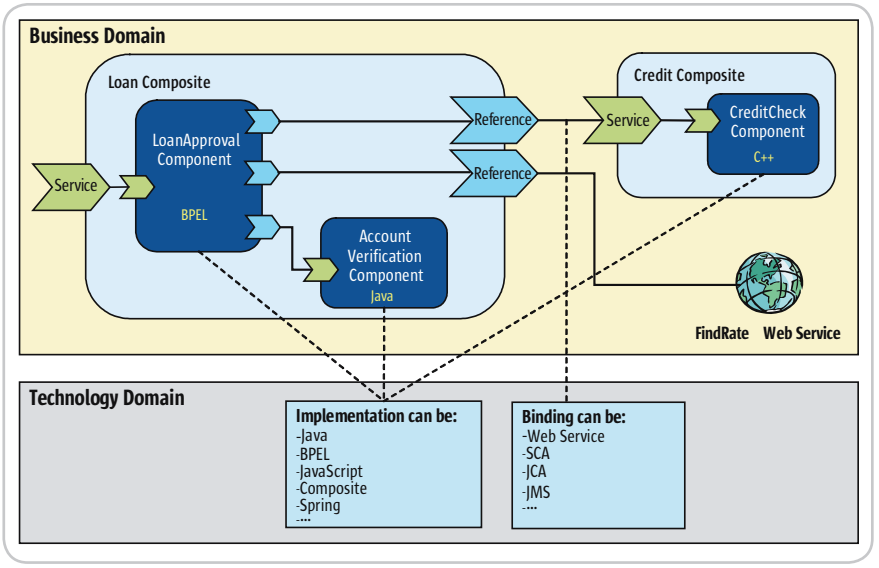


Figure 1 Loosely-coupled components in which business and technology domains are clearly separate

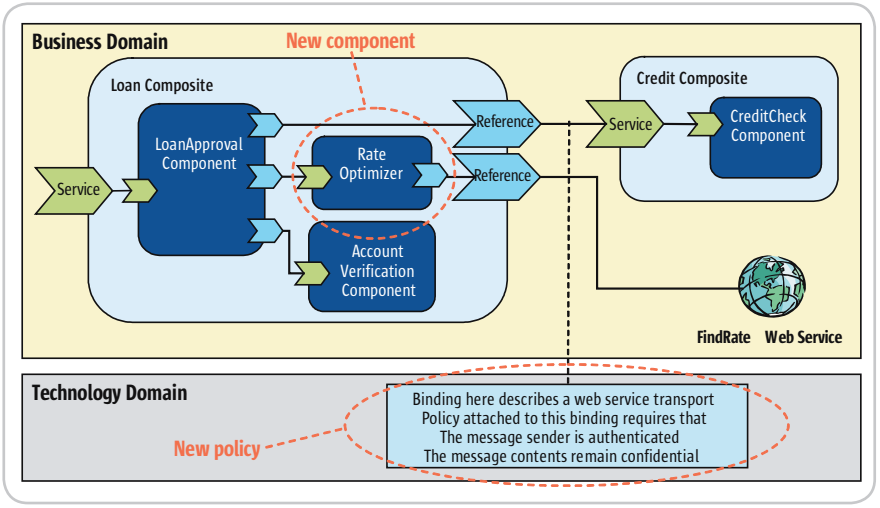


Figure 2 Updating the application: Binding policy and an additional component

"on ramp" for developers who want to create applications using a service-oriented approach. As an early implementer of SCA and SDO specifications, the Tuscany project is able to provide timely feedback on the specification to the Open SOA collaboration. Other implementations of this technology are also beginning to appear, for example, the PHP PECL SDO project at http://pecl.php.net/package/sca_sdo.

In summary, today's organizations must be able to quickly react to change. SCA promotes flexible and reusable solutions by encouraging componentization and by clearly separating business logic from underlying technology concerns. SCA and SDO independently increase developer productivity by shielding them from infrastructure complexity and the necessity to develop deep infrastructure technology skills. SCA and SDO together provide IT with a flexible model for building SOA based solutions and, more importantly, for effectively and efficiently handling change.

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

Oracle EDA Suite

Supporting events without complex custom coding

Reviewed by
Mark Simpson
and Mark Waite

An event-driven architecture (EDA) reflects the real world in which businesses operate. The real world is constantly changing, chaotic, and unpredictable. An EDA enables organizations to make sense out of all the events occurring within their business, and to detect anomalous business situations by drawing together a number of indirectly related or independent events. Furthermore, EDA builds decision-making capabilities directly into business processes by using analytical insights to drive decisions. EDA offers organizations the ability to track events in real time, thus gaining an early awareness of issues, improving productivity, and reducing manual intervention and errors.

Event-driven architecture is not a new application pattern; applications have been supporting events for years. What's new is that vendors are now enhancing software infrastructure products and application frameworks to support events without complex custom coding. This article reviews one of the latest products to do so – Oracle EDA Suite.

The Need For Event-Driven Architecture

Today, EDA is gaining popularity, driven by the need to solve the following business issues:

- Organizations are suffering from an overabundance of data resulting from the "double whammy" of rising transaction volumes and the increasing speed at which data is produced. A typical business may produce millions of events on a daily basis. Financial services firms often process up to 150,000 external events each second, a number that is forecast to increase to 5 million within three years.
- Customers have come to expect dramatically faster customer service response times – pushing organizations to react and respond faster than ever before – and in many cases customers demand proactive communication and resolution. At the same time, customer interaction channels are continuing to increase, requiring near real-time coordination.
- Over the years, businesses have created increasingly complex, heterogeneous IT environments. With Internet-linked, distributed systems, along with the rise in service-oriented computer systems, the complexity of systems has increased significantly, as chains of services distributed across multiple boundaries interlink in ways that will make administrators pine for the days of linear, procedural codes that ran on a single machine. Understanding what is happening at any given moment within such multifaceted applications is difficult, and this complexity will only increase.
- Regulations such as Sarbanes-Oxley require up-to-the minute reporting and end-to-end process visibility.

Organizations must sense and respond to events across the extended enterprise rather than carrying out predetermined processes. Organizations that incorporate event-driven styles into their enterprise architecture can respond more quickly to changing business conditions, whereas current infrastructures for processing and managing events require complex and expensive software engineering.

EDA or SOA? Why Another Architecture?

Developers using a service-oriented architecture (SOA)-based approach build an application by assembling "services," or software components that define reusable business functions. SOA is based on a conventional request/reply mechanism. A service consumer invokes a service provider by sending a message asking for some action or data, and then has to wait until the completion of the operation on the provider side.

Unlike the request/reply approach of SOA, where callers must explicitly request information, EDA allows systems to respond dynamically as events oc-

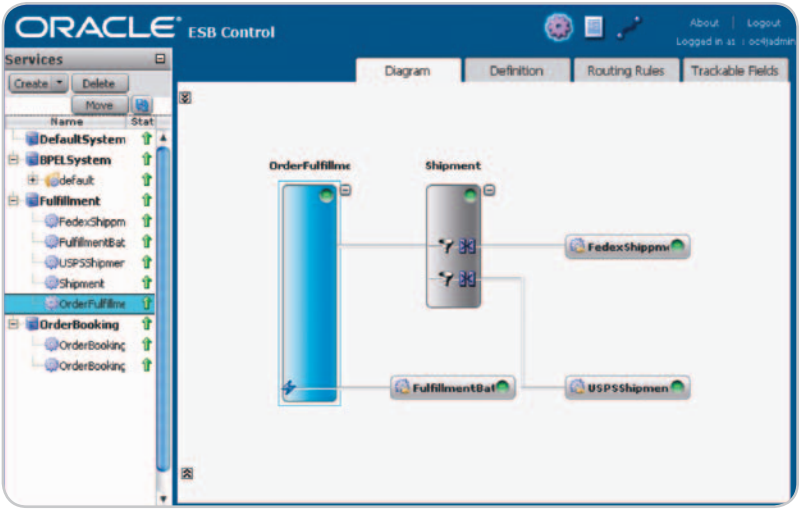


Figure 1 ESB Control Console for managing message routing

Mark Simpson has spent more than 12 years working with independent software vendors and systems integrators since graduating from Birmingham University, UK, with a BSC honors degree in Mathematics. Mark joined GW in 1998 he is now a Senior Solutions Architect responsible for strategic customers.

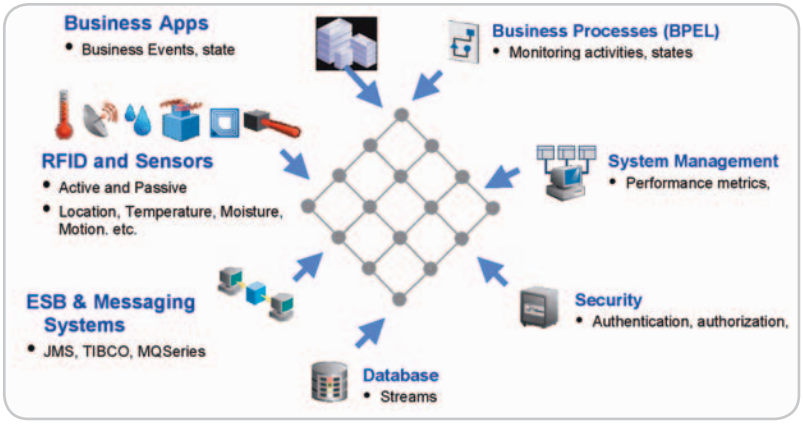


Figure 2 Heterogeneous events correlated for use in Oracle BAM

cur. In an EDA, event producers publish events, and event consumers subscribe to receive these events as they happen. Moreover, the generation of an event is not dependent on the availability of a service to process that event; events therefore must be stored to support such instances when a subscriber is not available.

Apart from its support for parallel asynchronous flows of data – in which information is transmitted without any anticipation of an immediate reply, and in which there is no need for a continuous connection between systems – event handling exhibits a number of other characteristics that serve to distinguish it from service processing.

Events require not only one-to-one exchanges, but also the additional capability for one-to-many and many-to-many communications. In addition, event handling allows the publisher and the subscriber to interact without knowing anything about each other. The relationship between the two systems is purely in terms of the information sent and received. In contrast, interaction between service components generally involves linear bidirectional request/response communications between a "client" and "server" service, with the flow controlled by the initiator.

EDA in the Real World

Real-world examples of event-driven processes include demand-driven manufacturing and dynamic pricing. In demand-driven manufacturing, production is initiated upon the receipt of a customer order, rather than building to a quota determined from past sales. This reduces inventory and enables each order to be customized: for example, Dell Inc. can manufacture to order within a 24-hour cycle.

In a dynamic airline pricing system, each new booking "event" triggers a new pricing calculation for the next buyer. Dynamic pricing maximizes revenue by charging more if demand is strong and less if demand is weak.

Solving use cases such as these requires going beyond traditional business process automation, to enable rapid response to changing conditions. This requires the use of both service and event processing, which are compatible and reliant on each other. Using events to wire together business processes enables more decision-making to be transferred from man to machine.

This interaction between EDA and SOA is two-fold. The occurrence of an event can trigger the invocation of one or many services. Those services may perform simple functions, or entire business processes. Secondly, a service may generate an event. The event may signify a problem or impending problem, an opportunity, a threshold, or a deviation. Upon generation, the event is immediately disseminated to all interested parties (human or automated). The interested parties evaluate the event, and take action if necessary. The event-driven action may include the invocation of a service, the triggering of a business process, and/or further publication or syndication of information.

SOA delivers software functions as loosely linked services that can be plugged, unplugged, or combined to form new applications. EDA enables organizations to respond instantly to any relevant event. It is clear that organizations need both SOA and EDA within their enterprise architectures.

EDA is not the successor to SOA; it is its sibling. Neither is it new: simple event-driven processing has been in common use for at least ten years with message-oriented middleware. However, one of the main advances in the past few years has been the emergence of higher-level programming tools and paradigms that make EDA implementation much less daunting.

The remainder of this article will focus on the requirements of an EDA and how Oracle EDA Suite supports those requirements.

Requirements of an EDA Solution

Successfully implementing an event-driven architecture requires the following components:

- ***Data and Event Collection Infrastructure:*** A real-time scalable infrastructure is required for event handling. Data is most valuable when it is fresh and can be acted upon immediately; stale data that cannot be used to anticipate events before they happen is of little value. The platform must provide both the scalability and reliability required to handle the vast amounts of real-time data that will be generated from EDA applications.
- ***Event Filtering:*** Not all events are of equal significance. It is important to be able to filter events – especially in high-volume cases such as RFID – to ensure networks and applications are not overwhelmed.
- ***Complex Event Processing (CEP):*** CEP deals with the task of processing multiple streams of simple events with the goal of identifying the meaningful events within those streams. CEP helps discover complex, inferred events by analyzing other events.
- ***Messaging Backbone:*** A messaging backbone that handles standards-based, multiprotocol messaging is required to support heterogeneous IT environments in which events are generated from a variety of sources and technologies, such as enterprise applications, databases, and RFID sensors.
- ***Visualization and Actionable Alerts:*** Users need to be able to visualize the events that are occurring within their businesses. They need to be able to quickly determine the nature of the problem, drill down into operational data, and take action directly.

Mark Waite is the co-founder of Griffiths Waite, a UK consultancy specializing in the delivery of enterprise architectures and composite applications. He has 20 years IT experience and holds a BSC honors degree in Computing Information Systems. He is currently responsible for the strategic direction of the company's enterprise applications practice.

Oracle EDA Suite

The Oracle EDA Suite consists of a subset of Oracle's middleware platform – Oracle Fusion Middleware – that allows customers to identify, analyze, and respond to business events in real time. Oracle EDA Suite can be implemented on its own or in conjunction with the companion Oracle SOA (Service-Oriented Architecture) Suite. Both share several overlapping components, including Oracle Enterprise Service Bus.

Oracle EDA Suite comprises the following components: Oracle Enterprise Messaging Service, Oracle Enterprise Service Bus, Oracle Business Rules, Oracle Business Activity Monitoring, and Oracle Sensor Edge Server. These components are described in more detail below.

Oracle Enterprise Messaging Service

Oracle Enterprise Messaging Service (OEMS) provides a standard messaging platform for EDA. OEMS is built on Java 2 Enterprise Edition (J2EE) standards such as the Java Message Service (JMS) and the J2EE Connector Architecture (JCA). For organizations that want to integrate their existing messaging technology with the Oracle platform, the OEMS JCA Connector implementation supports WebSphere MQ, TIBCO Enterprise JMS, and SonicMQ integration. PL/SQL and C APIs are also provided to allow integration with non-Java applications. In addition, Oracle Enterprise Messaging Service also leverages the security and high-availability features of Oracle Database and Oracle Real Application Clusters (RAC).

Oracle Enterprise Service Bus

Oracle Enterprise Service Bus (ESB) fully supports all the capabilities you would expect from a leading enterprise service bus: data transformation and document enrichment using XSLT or XQuery transformation, business rules, system cross-references, and domain value mapping. Oracle ESB provides connectivity by leveraging Oracle Adapters, which provide standards-based access to virtually any data source. Oracle ESB also supports content-based routing and content filtering. A novel feature is support for multiple protocols in the messaging bus, including JMS, SOAP, JCA, WSIF, JDBC, HTTP, and FTP. This means that organizations can get enterprise service bus capabilities – messaging, routing, and transformations – running on their choice of underlying protocol.

In contrast to Oracle BPEL Process Manager, which orchestrates long-running stateful business processes, Oracle ESB delivers high-performance messaging in support of both stateless and stateful service orchestration. Oracle ESB can capture the intermediate steps of a business process, allowing critical usage patterns to be identified and integrated into decision support systems in real time. Usage patterns can be identified to streamline business processes, or even head off problems before they cause irrevocable damage.

Oracle ESB combines event-driven and service-oriented approaches to simplify integration across heterogeneous platforms. It acts as an intermediary layer to enable communication between different application processes. A consumer or an event can trigger a service deployed onto Oracle ESB. It supports synchronous and asynchronous data flows, facilitating interactions between one or many stakeholders (one-to-one or many-to-many communications). Oracle ESB provides all the capabilities required for both service- and event-oriented architectures. (Figure 1 shows the ESB Control Console, which is used to manage messaging.)

Oracle Business Rules

Oracle Business Rules enables business analysts to easily define, update, and manage key decisions and policies governing business processes and applications.

Oracle Business Rules consists of the Rule Author tool, a Rules engine, and a Rules SDK. The Rule Author tool presents a simple interface for declaring rules that can be used by both programmers and business analysts. Rule Author generates the Oracle Rules language in a repository for use by the Rules engine. This language provides integration with Java programs, Web services, and XML documents. The Rules engine is a fast and efficient JSR-94 compliant RETE-based engine written in Java. The Rules SDK provides a rules-editing interface that allows applications to generate custom rules. The SDK is attractive for applications that define policies via their own special graphical interfaces. You can develop applications in Oracle BPEL Process Manager – part of Oracle SOA Suite – and capture business policies that are part of business processes by using the Rules engine. In this way, changes to business policies can be made without touching the business processes themselves. The Rules engine also supports integration with third-party engines such as iLog.

Figure 3 Oracle BAM dashboard supporting financial services scenario

Oracle Business Activity Monitoring

Oracle Business Activity Monitoring (BAM) enables companies to define and monitor events and event patterns that occur throughout their organization. Oracle BAM captures information from custom and packaged applications; business processes and workflows; databases; messaging systems such as JMS, AQ, MQ; and other systems to collect data in real time. Oracle BAM is also fully integrated with Oracle BPEL Process Manager to collect process information in real time. (Figure 2 shows the diverse nature of events supported by Oracle BAM.)

The Oracle BAM architecture delivers requested critical information within seconds of an event or change in status. Because the primary source of data is messages, Oracle BAM can update reports and generate alerts at speeds that traditional architectures can't match. Oracle BAM can accept tens of thousands of updates per second into a memory-based persistent cache that is at the center of the Oracle BAM architecture.

Users can view dashboards and reports showing critical business measures and key performance indicators (KPIs) that update in real time, and can then drill down into the detailed information underlying them. Dashboards automatically update as new events occur. Only events that effect changes to the dashboard display are sent across the network, and a connection is left open from the dashboard to the Oracle BAM server to receive these updates, removing the need for time-consuming polling and reducing server load.

Oracle BAM allows business users to be alerted when business conditions are out of band. Alerts can be delivered on a variety of devices. Alerts are fully actionable, and can invoke external programs or Web services to change the underlying business operation in real time. Users can take any necessary corrective action on monitored events – for example, launching a business process in Oracle BPEL Process Manager – right from the dashboard.

Oracle Sensor Edge Server

Oracle Sensor Edge Server provides the link to the physical world, enabling organizations to collect and manage data from sensors placed strategically around the enterprise, and observe what is happening in the world outside of IT. Oracle Sensor Edge Server captures data from any sensors, RFID equipment, or other external devices and publishes it to enterprise applications, while also relaying instructions to response devices such as light stacks, printers, and other material handling equipment. The captured data is normalized to ensure consistency between sensors, and then filtered to prevent "event flooding" by reducing the amount of data that needs to be handled by the network and applications. The data can then be routed to the appropriate applications through Oracle ESB, or visualized in the Oracle BAM dashboard.

Business Scenario: Financial Services Company

To fully assess the capabilities of Oracle EDA Suite, it's helpful to model a real-world scenario that captures a typical business process. This scenario is a high-volume case within the increasingly competitive market of financial services loan brokering. The organization is an intermediate broker that takes loan requests from multiple sources and sells the requests to suitable lenders with varied and flexible commission structures. It is a start-up company that operates nationally, with aggressive growth plans within its domestic U.K. market and in Europe, the U.S., and Asia. Being new to the industry, with tight reins on budget, the company must closely monitor the cost of acquiring leads in relation to the revenue generated by those leads. With limited funds for marketing, the company must utilize all resources to full capacity.

The company used Oracle EDA Suite as the foundation of its brokering system to quickly gain a foothold in the market and achieve its goal of being the first broker to offer customers a deal, thus removing them from the market. By using Oracle Enterprise Messaging Service (OEMS), the company receives credit leads from a large number of diverse sources. The initial lead generation was done through a number of Websites that each target a different segment of the market. The Websites produce XML messages that are transported to the brokering system via an in-memory JMS implementation.

OEMS validates and authenticates the data before placing the messages on the Oracle Enterprise Service Bus (ESB) to securely route the loan applications in different formats to the most suitable lenders, dependent on rules configured in Oracle Business Rules. Together, OEMS and Oracle ESB not only provide seamless integration to lenders who use other messaging systems, such as WebSphere or Sonic ESB, but also allow persistence of application data while manual underwriting is performed by lenders who do not have automated systems in place.

Oracle ESB and OEMS give the broker the necessary transaction support to handle exceptions with third-party systems, and Oracle BAM offers underwriters visibility into the state of any application. Oracle BAM gives the broker a complete view of the state of operations through role-based dashboards that combine real-time lead information with current workload data and data on the current and historical performance of lenders to ensure that the business rules used by Oracle ESB to route applications are accurate and efficient, and the best service is provided to customers and lenders. (Figure 3 shows an Oracle BAM dashboard used to monitor lead generation.)

To support leads from nonautomated or partially automated sources, OEMS can receive bulk files of turndowns from other lenders in the form of e-mails or

Criteria	Description	Average Score
Capability	All Components Required for EDA	9.5
Interoperability	Support of all Major Standards	8
Ease of Use	Tutorial Driven Graphical Font Ends	8.5
Infrastructure	All Grid Benefits of Oracle Application Server	9
Installation / Administration	ESB Console a useful addition	8
Overall	**A fully rounded EDA suite**	**8.6**

Table 1 Design Time

> "It is clear that organizations need both SOA and EDA within their enterprise architectures. EDA is not the successor to SOA; it is its sibling. Neither is it new: simple event-driven processing has been in common use for at least ten years with message-oriented middleware"

direct communications. In addition, individually sourced messages from partner systems are accepted by OEMS. This receipt of heterogeneous messages from diverse brokers gives the start-up company a great advantage: it can rapidly scale up in lead volume, and quickly sign up a large number of partner brokers and lenders, meeting their technology requirements rather than enforcing an integration protocol.

The complex event processing (CEP) within Oracle BAM correlates the applications received from partners with messages from elsewhere in the business that record events such as SLA warnings from low-performing lenders, opportunity thresholds with efficient lenders, or activity triggered from market analysis. The CEP engine allows the broker to work with flexible commission models, leading to preferred lending rates and faster acceptance of leads. The broker is anticipating making even more investment in the Oracle CEP engine when it becomes a standalone product with a business front end: the marketing department will utilize the tool to filter, correlate, and aggregate application events to identify cross-sell and up-sell opportunities such as upgrading the loan or transporting events to partners focused on offering mobile phone contracts or car loans.

One of the main sources of leads for the company is links, sponsored by Website search engines that present qualified traffic. Each day the broker goes through an auction process to buy search engine keywords such as "Low APR" or "Debt Consolidation." Search engine marketing is a very effective and efficient way of increasing revenue for the broker: it is more adaptable, scalable, and responsive than traditional advertising, giving the broker the ability to turn it off, up, or down in real time.

However, in the consumer credit business, brokers run the risk of a lot of browsing traffic but few buyers. Because the cost of keywords is click-based, it is essential that the volume and benefit of leads from this channel be closely monitored. Oracle BAM analyzes the volume and cost effectiveness of these clicks, tying the clicks back to account performance to determine the value of the advertisements. Thresholds are set up to cancel bad keywords immediately and transfer the marketing budget to other keywords.

Oracle BAM correlates the events streamed from these search engine clicks with the performance of the generated leads, enabling the broker to optimize its search engine marketing campaigns in real time, and giving it confidence to invest valuable budget in this lucrative but risky method of lead generation. The advertising can be as responsive to the brokers' events as required: when Oracle BAM shows that current performance is low, the search engine marketing can be scaled up, resulting in an immediate response.

The scalability and comprehensive support of messaging systems within Oracle EDA Suite has given this start-up broker the infrastructure required to achieve its goals of being the number one lead generator in its territory and expanding globally in its vertical market. Just as important, it allows the broker to handle more than just loan applications, and become the central hub as a data broker across many markets without any technology barriers.

Conclusion

Oracle EDA Suite has several key benefits. First, all the products within the family have been engineered to work together, which provides out-of-the-box integration benefits. Implementation times are faster, which in turn means quicker time-to-business benefit. Companies do not have to spend time setting the environment up, but can get straight down to business, concentrating on satisfying their business objectives instead of sorting out the plumbing.

Second, Oracle Fusion Middleware, because it is based on industry standards, is highly interoperable. Products in the family can work with third-party middleware products and databases, as well as across heterogeneous business applications. This is particularly important to a consultancy company that has to work with what organizations already own – applications, technology, and infrastructure from any vendor. With Oracle EDA Suite, there's no need for the client to rip out and replace what is currently there and already working. It can leverage its previous technology investments, thus significantly reducing the cost and speed of implementation while also minimizing disruption and ultimately business risk.

Oracle EDA Suite and Oracle SOA Suite are both components of Oracle's next generation of service-oriented architecture that defines how events and services are linked together to deliver a flexible and responsive IT infrastructure. We agree that event processing is an emerging requirement that will increasingly find its way into more and more enterprise applications. Without the interoperability with applications provided by event-driven SOA, an EDA solution will be compromised, as the solution can only be as good as the events generated from the organization's business applications and Web services.

Oracle EDA Suite provides organizations with all the key building blocks they need to event-enable their IT infrastructure, and we strongly recommend the suite to organizations considering such a solution. However, Oracle's most compelling proposition is the ability to deploy Oracle EDA Suite in conjunction with Oracle SOA Suite to satisfy both event-driven and service-oriented infrastructure requirements in a single integrated platform.

Rating

We rated Oracle EDA Suite on the criteria (see Table 1), with assessment from architects, developers, and administrators.

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

Parasoft Jtest 8.0

A real heavyweight

Reviewed by
Jason Bell

In terms of unit testing and code compliance, Jtest is a real heavyweight in the arena. For those who haven't come across Jtest before, it's an application that will analyze your Java application code for you. At present Jtest has 700 built-in rules and 100 security rules and it will autocorrect 250 of those rules for you. It provides Parasoft SOAtest hooks for testing of SOA/Web services and Web apps. The reporting engine is also built-in so once tests are run, you can view and print results via a Web browser. There are some new features such as improved J2EE testing and the Bug Detective, which I will cover later in this review.

The front end is built on the Eclipse framework so it will be familiar to some of you. Test projects are created the same way you would create a project in Eclipse. The wizards are easy to use and I got up and running in a short time. You can also buy Jtest as a plug-in for Eclipse (versions 3.0 – 3.2) and IBM Rational Application Developer. I would strongly suggest playing with the example projects for a short time before using your own code, just to get used to how the application operates.

Basic Testing

Once a project is set up, testing the code is a fairly simple matter. Don't try and test every Java file in one huge test. Jtest is quite memory intensive; my Pentium4 1GB laptop decided to die as the machine couldn't handle all the processing. Parasoft recommends 2GB of memory to use Jtest properly. Selecting a couple of files or a package at a time usually works best. There are built-in configurations to test certain aspects or specific texts written in the past (there's a configuration based on Joshua Bloch's "Effective Java" series, for example). You can configure Jtest to run as many or as few of the rules as you want. A good starting point is to check for standardization guidelines. Jtest will also create JUnit test cases in JUnit 3 format and now includes execution of JUnit 4 test cases. In one click you can standardize your code, write the test cases and execute them, then finally read the report to see how it all went. There is a tabbed review of the tests run and their outcomes; highlighted problems are easily found – it's a matter of double-clicking the error. From there the QuickFix tool can correct most problems that are found.

Team Working

If you work in a team, it's pretty safe to assume that you would be using some form of source code control. Jtest can work with CVS, Subversion, ClearCase, and StarTeam. Jtest has improved the team-working aspect and can monitor commits and check-ins from defined team members. This is handy for developers in different countries; let's assume there is a team manager and a small group of developers. The manager can now arrive at work in the morning, launch Jtest, and review all the latest CVS-committed code by the team members. From the CVS logs, Jtest can tell you what has been changed

Parasoft Corporation

101 E. Huntington Drive
Monrovia, CA 91016
Web: www.parasoft.com
E-mail: info@parasoft.com
Phone: 888-305-0041

Jason Bell is founder of Aerleasing, a B2B auction site for the airline industry. He has been involved in numerous business intelligence companies and start ups and is based in Northern Ireland.

jasonbell@sys-con.com

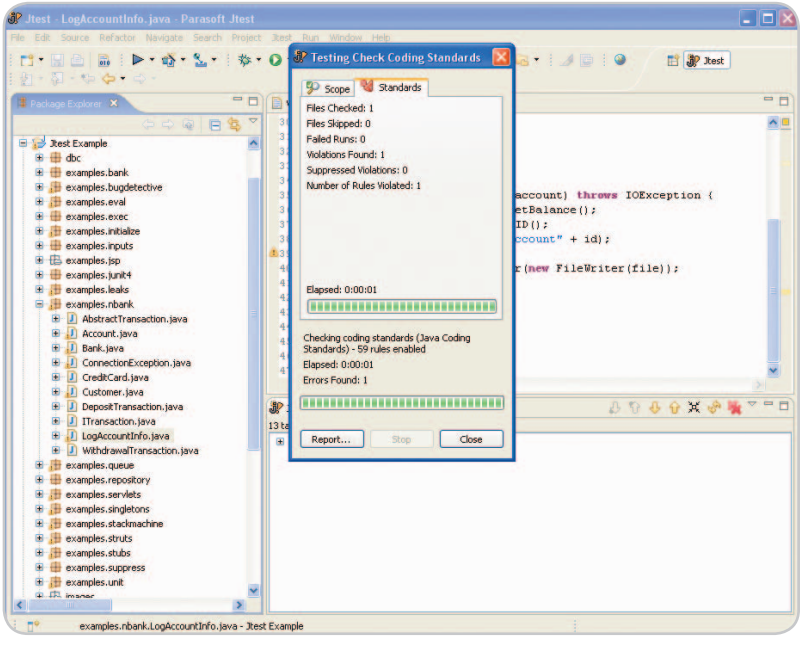


Figure 1

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

> Jtest supports Struts, Spring, and Hibernate projects and also the standard J2EE EJB/JSP and servlet specs"

and how the tests have performed. Any corrections can be noted via Jtest and the team members will be notified when they start their working day. In theory, all the developers have tested the code and all the developers have notes via the CVS log on any corrections or complications that need addressing. This does assume that all team members are using Jtest. For a large corporation with many developers, this is a serious benefit. Over time, you can see how the code quality and test coverage has performed.

J2EE Testing

Jtest supports Struts, Spring, and Hibernate projects and also the standard J2EE EJB/JSP and servlet specs. These items can also go through the same rigorous standards testing as all other types of source code. Some of the code convention rules have been designed for the likes of Struts and Hibernate applications.

BugDetective

Testing applications prior to execution is all very well but the majority of bugs only surface when running the application. We can run our own test harnesses but these usually scratch the surface of what actually needs testing. In Jtest 8 there is a new feature – BugDetective, a new static analysis technology that simulates execution to automatically identify real execution paths – often paths that span multiple methods, classes, and/or packages – that will lead to runtime bugs such as NullPointerExceptions, resource leaks, SQL injections, and other security vulnerabilities. This is helpful in terms of database access applications as it's difficult to establish what the problems are going to be. It also means that GUI applications can be tested properly. During my testing, no matter what I threw in, in terms of bad coding (NullPointerExceptions, not closing open I/O streams, etc.), the BugDetective managed the whole thing with flying colors.

System Requirements

Operating System

- Windows: Windows 2000, XP, or 2003
- Linux: Red Hat 9.0, Fedora Core 1-3 or higher, Red Hat E.L. 2,3
- Solaris: Solaris 9 or 10

Hardware

- Intel Pentium III 1.0 GHZ or higher recommended
- 512MB RAM minimum; 2GB RAM recommended
- Sun Microsystems JRE 1.3 or higher

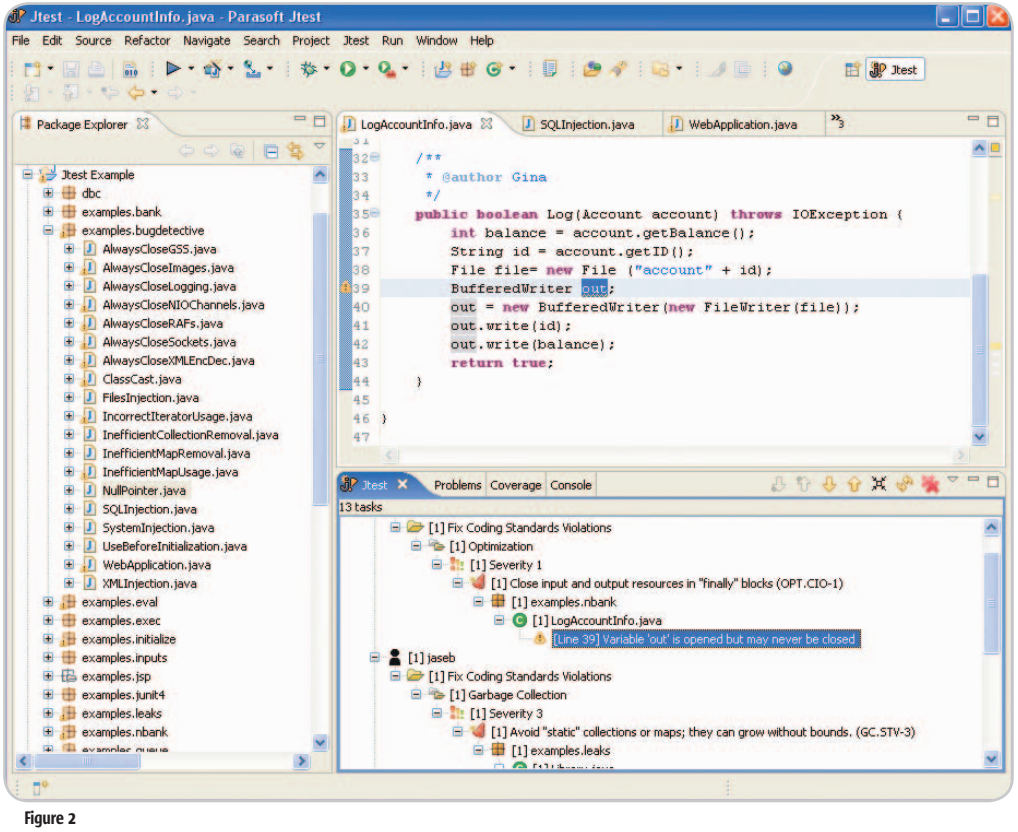


Figure 2

Jtest Tracer

Bug Detective has one more feature up its sleeve: the ability to write JUnit test cases on the fly while your program is running. In the example source, there is a basic GUI application with a LIFO and FIFO stack. While the program is being run and the user enters some test data Jtest is putting together a test case in the order in which it was run. It's an achievement that should not be left on the side. It may be a new feature but this brings Jtest into a class of its own.

Conclusion

Jtest has matured with age and is getting like a fine wine now. There are facets that I discover every day that make me open my eyes and go, "Wow!" It's a large complicated product, but for a large team it's essential. This product is not for everyone and not every developer has the cash for the license fee. I'd love to see Parasoft do a stripped-down community version of Jtest, even if it was a stripped-down version of the coding rules and tests.

JAVA DEVELOPER'S JOURNAL Advertiser Index

Advertiser	URL	Phone	Page
AjaxWorld East Conference 2007	www.ajaxworldexpo.com	201-802-3022	49
Altova	www.altova.com	978-816-1600	4
Backbase	www.backbase.com/jsf	866-800-8996	21
Business Objects	www.businessobjects.com/devxi/misunderstood		11
Cynergy	www.cynergysystems.com		31
IBM	ibm.com/takebackcontrol/flexible		8-9
ICEsoft Technologies	www.icesoft.com	877-263-3822	25
Infragistics	www.infragistics.com/jsf	800-231-8588	13
Instantiations	www.instantiations.com/rcpdeveloper/resources/casestudy-bea.pdf		32-33
InterSystems	www.intersystems.com/jalapeno2p	617-621-0600	7
IT Solutions Guide	www.itsolutions.sys-con.com	888-303-5282	61
Java Developer's Journal	www.jdj.sys-con.com	888-303-5282	57
Laszlo	www.openlaszlo.org		43
Northwoods Software Corp.	www.nwoods.com	800-434-9820	47
OPNET Technologies, Inc.	www.opnet.com/pinpoint	240-497-3000	15
Parasoft Corporation	www.parasoft.com/jdjmagazine	888-305-0041	Cover IV
Quest Software	www.quest.com/hero	949-754-8000	Cover II
Software FX	www.softwarefx.com	800-392-4278	Cover III
SYS-CON Website	www.sys-con.com	888-303-5282	59
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.

JSR 306 Gets Noticed, Draws Valuable Feedback

Improving the JCP

Onno Kluyt

Our new effort to improve and change the Java Community Process through JSR 306 is still young; however, developers and all those interested have already started to provide valuable feedback and share their opinions generously. One such place where opinions were expressed early was the poll on JCP change that the java.net site put up (http://today.java.net/pub/pq/123). "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. Acknowledging the comments provided by some of the voters in the poll, it serves us to look at the process of joining the JCP, making the expert group discussions more visible, and allowing members and non-members alike to participate in these. One commentator states that the JCP should look closely at various open source projects and accept some as standards. Artima.com reported on JSR 306 and readers there commented on the openness and transparency of discussions. The Java Posse took notice of the JSR as well, asking questions and requesting more information about the item on allowing non-Java implementations of some of the specs.

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 indeed 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 reference implementation and a technology compatibility kit). This makes the membership agreement complex to a degree that individuals are not used to dealing with. The Website (JCP.org) can 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.

Openness and the transparency of expert group discussions and other related communications at the JCP are topics that are frequently raised and they are very valid ones. From a developer's point of view, it's difficult to understand why public access is not granted on Java specification efforts that the developer is interested in. It is also difficult to explain! In earlier versions of the JCP, the first draft review, then known as the 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 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.

The accessibility of expert group discussions has, of course, a direct relationship to the perception of involvement as experienced by members and by potential new members. While JSR 306 will bow over to the various legalities and process regulations that may be at play, involvement also has a strong usability aspect. This is an area where the Website can play a role. Since JCP 2.6, spec leads have had the ability to post whatever updates they want to share (notes, working drafts) with the community as a whole. These are the so-called Community Update pages for each JSR. How do you become aware of these pages? One option is to track your favorite JSRs and RSS feeds through Website features that enable users to do just that.

One other thing the process-change JSR sets out to explore and implement is allowing for non-Java implementations of specifications created by some of the JSRs. In EC talk, these have been labeled as "Hybrid JSRs." Sometimes our nicknames are a tad more fanciful. There is something called "Purple JSRs" but that' a different story. By "non-Java" we 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 in which 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 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 a different world than Java.

As always my expert group and I are very interested in your views, keep them coming. You can send your comments to me directly (onno@jcp.org) or to the expert group (jsr-306-comments@jcp.org).

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

onno@jcp.org

Eclipse Data Visualization

(No Silly Glasses Required)

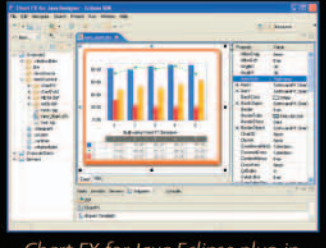


Chart FX for Java Eclipse plug-in.

The Leading Charting Solution Now Provides Powerful Data Visualization for Eclipse

The **Chart FX for Java 6.2 Eclipse plug-in** brings enterprise-level data visualization features to the Eclipse IDE. The Designer is integrated into the IDE allowing quick customization of the charts and the required code generation. In addition to a myriad of traditional chart types, the **Chart FX Maps** extension is included to create dynamic, data-driven image maps, such as geographic maps, seating charts or network diagrams, among others. Chart FX for Java 6.2 is available as a Server-side Bean that runs on most popular Java Application Servers. The 100% Java component produces charts in PNG, JPEG, SVG and FLASH formats. The **Chart FX Resource Center** integrates into the Eclipse Help and includes a Programmer's Guide, the Javadoc API and hundreds of samples. This makes Chart FX for Java the most feature-rich, easy-to-use charting tool available for Java development. *Learn more about the seamless integration and powerful features at www.softwarefx.com.*

Chart FX

www.softwarefx.com

New! version 6.2

Now Includes Maps!

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