AJAX Is Here to Stay

JAVAPro™

www.javapro.com A Fawcette Technical Publication 2005/2006 Vol. 9 No. 6

The Jini Solution to SOA

Build an Infrastructure for Federating Services

Migrate Apps to EJB 3's Business Logic Model

Resolve Enterprise JDBC Connection Issues

History Can Dictate Your Tooling Approach

Retrofit a JFace-Built UI to an Eclipse Plug-In

Satisfy Legacy Requirements without Upgrades

WebLogic Integration for Managing Human Workflows

Enterprise Architect Summit 2006

Strategies and Best Practices for the Real World

Enterprise Architect Summit returns to Florida in May for three informative days of keynotes, workshops, and breakout sessions led by experts in the enterprise architecture field. Arm your business to respond to emerging IT challenges – register today.

The Ritz-Carlton Key Biscayne Resort, Florida

May 15-17, 2006

Sessions Will Cover:

- Putting the A in SOA
- Solving Real-World Architecture Issues
- Building an Agile Enterprise
- Key Strategies to Implement Security Policies
- Metadata: the Key to Understanding IT
- Joining Enterprises with the Global SOA

Register by March 22 and save $300.

Call 800-848-5523 today

or visit us online at

www.enterprise-architect.net/summit

FTP FAWCETTE TECHNICAL PUBLICATIONS

Take Your Business to the Next Level-

View the preliminary list of sessions, workshops, and events planned for May

Monday, May 15

8 a.m.-6 p.m.	**Registration**	
8 a.m.-3 p.m.	Enterprise Architect Classic Golf Tournament	
	Workshops	
1-5 p.m.	**Best Practices:** Security Policy Development and Enforcement	**Strategy:** IT Drivers for Business Process Management
6-8 p.m.	Welcome Reception	

Tuesday, May 16

9 a.m.	**Keynote**	
	Best Practices	**Strategy**
10:30 a.m.	Putting the A in SOA	Solving Real-World Architecture Issues
11:45 a.m.	Best Practices for Database Connectivity in the Midst of Infrastructure Diversity	The Enterprise Architecture Office and the Ever-Increasing Organizational Need
12:45 p.m.	Lunch	
2 p.m.	**Keynote**	
3:30 p.m.	The Rocky Road to Compliance	*Panel:* Key Strategies to Implement Security Policies
4:45 p.m.	Building an Agile Enterprise	Metadata: The Key to Understanding IT
6 p.m.	Exhibitor Reception	

Wednesday, May 17

9 a.m.	**Keynote**	
	Best Practices	**Strategy**
10:30 a.m.	Model Driven Software Engineering	Joining Enterprises with the Global SOA
11:45 a.m.	*Panel:* Perspectives on Architecture Modeling	The Data-Centric Enterprise: A Blueprint for Enterprise Architecture
12:45 p.m.	Lunch	
2 p.m.	Managing Dependencies Across the Architecture	Selecting SOA Infrastructure
3:15 p.m.	Minimize Business-Disruption Risk and Cost Through Architectural Modeling	SOA Operations and Governance to Move Applications to the Network Architecture
4:30 p.m.	Build an Enterprise Security Architecture	SOA Models and Methodologies

JAVA Pro™

COVER STORY

FEATURES

DEPARTMENTS

COLUMNS

Java Pro (ISSN: 1096-4495) is published bimonthly with two additional special issues in June and October for the basic rate of $34.97 per year by Fawcette Technical Publications Inc., 2600 South El Camino Real, Suite 300 San Mateo, CA USA, 94403. Tel. (650) 378-7100; fax (650) 570-6307. **POSTMASTER:** Send address changes to ***Java Pro***, P.O. Box 3485, Northbrook, IL 60065-9819. Periodicals Postage Paid at San Mateo, CA and at additional mailing offices. **Canadian CPM: 40013204; Customer: 2418150; Return Address: 4960-2 Walker Road, Windsor, ON 49A 6J3, Canada. ISSN: 1096-4495.**

PUBLISHER, Jeff Hadfield

EDITORIAL
EDITOR, Terrence O'Donnell
TECHNICAL EDITOR, Daniel F. Savarese

CONTRIBUTORS
Kevin Jones, Daniel F. Savarese, and Peter Varhol

ART & PRODUCTION
VICE PRESIDENT, ART & PRODUCTION, Michael Hollister
SENIOR ART DIRECTOR, Bruce Gardner
PRODUCTION MANAGER, Kathleen Sweeney Cygnarowicz
ART DIRECTOR, Brian Rogers
ASSOCIATE WEB PRODUCER, Shane Lee

ADVERTISING SALES
EXECUTIVE ASSISTANT TO THE VICE PRESIDENT, PUBLISHING, Susan LaCroix
REGIONAL ACCOUNT MANAGER, Robyn Johnson

CIRCULATION
SENIOR CIRCULATION DIRECTOR, Karen Koenen
CIRCULATION MANAGER, Fred Perry
CONFERENCES ASSOCIATE, Gerry Guzman

MARKETING
MARKETING MANAGER, Susan Ogren
SENIOR DESIGNER, Margaret Horoszko

CONFERENCES
VICE PRESIDENT, Tim Smith
SALES OPERATIONS MANAGER, David Seymour
CONFERENCE OPERATIONS PLANNER, Will Hansen

CUSTOMER SERVICE
CUSTOMER SERVICE REPRESENTATIVE, Jose Porcell

OPERATIONS
EXECUTIVE VICE PRESIDENT/CHIEF FINANCIAL OFFICER, John Sutton
SYSTEM ADMINISTRATOR, Tin Cao
DIRECTOR OF FINANCE, Darlyn Phillips
ACCOUNTS PAYABLE ACCOUNTANT, Elena Ostrovsky
STAFF ACCOUNTANT, Betty Tsang-Hwah Wu
ACCOUNTS RECEIVABLE, Iain Niellands
HUMAN RESOURCES MANAGER, Pam Davis

FTPONLINE
MANAGING EDITOR/BUSINESS UNIT MANAGER, Nina Goldschlager
ASSOCIATE EDITOR, Lauren Dresnick
ADVERTISING DIRECTOR, Roy Kops
E-ADVERTISING MANAGER, David Seymour
EAST COAST SALES MANAGER, Dennis Leavey
WEST COAST SALES MANAGER, Lisa Sidlow

FAWCETTE TECHNICAL PUBLICATIONS
PRESIDENT, James E. Fawcette
VICE PRESIDENT, CHIEF INFORMATION OFFICER, Aaron Weule
VICE PRESIDENT, FTP NETWORK, Paul Spyksma
VICE PRESIDENT, PUBLISHING, Jeff Hadfield
CORPORATE COUNSEL, Wilson, Sonsini, Goodrich & Rosati

JAVAPro online

The ONLY place on the Web for *Java Pro* content, code, and community is www.javapro.com

Visit *Java Pro* online for this extended content and more.

FTPOnline Blogs

Check out the FTPOnline blog page to get insights from Jeff Hadfield, vice president, publishing, Terry O'Donnell, *Java Pro* editor, and other FTP editors sounding off on IT and development issues.
www.ftponline.com/weblogger/

Delivering on Big Thoughts

Locator+ code: PV_051022

HTTP://WWW.FTPONLINE.COM/WEBLOGGER/FORUM.ASPX?ID=11&DATE=10/22/2005&BLOG#444

by Peter Varhol

"In what we call an information society," said Peter Varhol in a recent blog, "the one thing we seem to lack is adequate information on which to make intelligent decisions." And he claims that despite our expectations from technology, this situation isn't any different than in past eras. Check out Peter's take on the heroic significance of a colleague's efforts to utilize data to help deliver a useful information-spreading application and why it exemplifies "big thought."

A Look at the Latest Java Studio Creator

Locator+ Code: TO_051031_1

HTTP://WWW.FTPONLINE.COM/WEBLOGGER/FORUM.ASPX?ID=13&DATE=01/26/2006&BLOG#519

Sun Microsystems is zeroing in on corporate developers with its next release of Java Studio Creator, an IDE that provides a rapid, visual, drag-and-drop Web application building environment that's a lot like the visual environment in Visual Basic or ASP.Net. See what else Terry O'Donnell, *Java Pro* editor, gleaned about Creator at the recent 2005 TopCoder Open in Santa Clara, California.

More Online Exclusives

Improve Access to Externalized Text

Locator+ code: JP051018CO_T

by Charles L. Owen

Separating text—error messages, widget labels, and help dialogs are typical examples—from source code without proper controls can make it difficult to ensure that source code remains correct. Such separation is often necessary to enable localization and ease efforts to keep text crisp, consistent, and correct. Take an approach that applies rigorous controls for efficient code.

What Are Locator+ Codes?

Locator+ codes give you instant access to a feature on *Java Pro* Online. Simply type the Locator+ code into the field in the upper-right corner of the page, and click on the "go" button.

JAVA INSIGHT NEWSLETTER SIGN-UP

www.javapro.com

Every week, the *Java Insight* e-mail newsletter brings you up-to-date news, technical information, opinions, interviews, and analysis on topics and technologies such as J2EE, middleware, and application servers; J2ME, devices, and embedded development; data access; servlets and JSP; Web services; and/or XML. Sign up for *free* at www.javapro.com.

Making a Connection

by Terrence **O'DONNELL**

Recently there have been opportunities galore to digest a tremendous amount of information about wireless and network connectivity and the many ways to leverage that connectivity. Whether through applications on handheld devices, laptops, or even desktops, connectivity is reshaping our world in innovative ways, and a lot of new technologies either just released or on the horizon are quite impressive.

One source of this information glut was an immersion into the cutting-edge technologies for mobile handheld devices exhibited at this year's Nokia Mobility Conference (NMC) in Europe, where announcements and buildup prior to and during the event dominated the wires this fall. Nokia's new S40 3rd Edition and S60 3rd Edition platforms (the latter built on top of the new Symbian OS 9.1, also marking a significant upgrade); a parade of sexy new devices that expand Nokia's E and N lines and sport a cornucopia of features in addition to advanced telephony; and the Carbide family of development tools were among the more splashy eye-openers from the global communications giant and its developer site, Forum Nokia.

Other industry events earlier in the year featured keynotes by Sun Microsystems' Scott McNealy, CEO, and Jonathan Schwartz, president, who both spoke about the ways in which the network and connectivity are moving the world forward—and McNealy's featured an appeal to reduce and perhaps one day eliminate the chasm existing between those who can get connected and those who can't. Schwartz equated the significance of this current shift to that of other historical milestones that have reshaped our societies, such as the invention of electricity and the industrial revolution.

What do these announcements and optimistic prognostication over connectivity mean to the Java community of IT professionals, architects, and developers? I pondered this question while synthesizing all of this material in light of what it means for the industry we cover. Despite some of the socioeconomic and political hurdles that will need to be dealt with to realize some of the loftier goals these technologies promise, it seems clear that we've embarked on a new movement that will give IT professionals an even wider highway for innovating the ways in which consumers and businesses will leverage connectivity.

The recent developments from Nokia provide relevant context. The S60 3rd Edition platform with its MIDP and CLDC support, new APIs, and enhanced platform security will open up greater opportunities for customization, according to Nokia's Matti Vänskä, vice president, mobile software sales and marketing, technology platforms.

"I think traditionally people have thought about customization as being a look-and-feel type of thing. Cosmetics," Vänskä said. "What we're doing with XML is decoupling the application functionality—what used to be done with C++—and what it looks like. We have the functionality that remains to be done, but then we have the kind of look-and-feel elements that are done with XML that allow completely different kinds of people to get involved in terminal design and terminal customization."

With that decoupling, said Vänskä, Nokia is introducing an opportunity where customization, on a deeper level, is about creating service propositions and application and service development. Developers are therefore in a key position because operators will increasingly want to leverage what's out in the market, and no single operator can define what the consumers want and has the capability to create it.

These opportunities expressed by Nokia executives paralleled messages reflected at JavaOne and BEAWorld by Sun's execs. At the latter event Schwartz stated that there are all kinds of wonderful services and infrastructures out there, and they indicate a definitive shift in the way that people are and will be connected.

"In that connection there will be opportunity," Schwartz told those attending his keynote. "And what was evident to us [Sun] was that *opportunity* in connecting people together has to be bigger than just one company. It has to be about bringing together communities to agree on standards so they can evolve them, and one of the shining aspects on the network today, which continues to evolve … is the Java community. Whether on handsets, on Blu-ray DVDs, in servers, or in automobiles, we're seeing it evolve in the most interesting ways possible. It continues to gain and grow in value … because we have a community that can come together with inspiration and innovation and go and drive it to the next level."

Java Pro would like to assist you in taking innovation to the next level. In the coming year look for more coverage in wireless, open source and standards, and other evolving technologies for Java application development and enterprise architecture. If you'd like to communicate *your* ideas and inspirations to the community through *Java Pro*, then let's get connected. JP

Terrence O'Donnell, Editor
todonnell@fawcette.com

From the Editors of *Java Pro*:

JAVAinsight™

FREE Weekly E-Mail News

How Much Java™ Do YOU Want?

Get the best in Java news, resources, how-to tips and more delivered to your inbox every week—FREE. With *Java Insight*, it's easy to keep up on the latest Java news and information.

Delivered every single week to your inbox, it gives you regular, spirited coverage of hot topics like:

- The latest in J2EE technology
- Using Java to develop Web services
- Creating Web apps with JSP
- Extending Java to embedded and wireless systems and devices
- And much, much more!

It's FREE, it's EASY, and it's chock full of JAVA!


Write Faster JDBC Applications

JAVAinsight
Vol. 4, No. 5, Febr. 2, 2005

Now featuring tips from jGuru

In This Issue:
- Security and the Custom Client
- Dispelling 10 ESB Myths
- Book Chapter Download: Java 2, JDK 5
- IT Job Market is Either Booming or Collapsing
- Determine Your SOA Readiness
- Nextel, RIM Offer Up Challenge

Security and the Custom Client
Though unconventional, a customized client solution is growing more popular for enterprises as the pressure mounts to increase the security of applications and client data. See why. [Read More]
- The Java Security Landscape
- Mapping Security to an SOA
- XML and Web Services: Are We Secure Yet?

Dispelling 10 ESB Myths
Misconceptions have arisen about what ESB is and what it's good for. Software Architecture Summit speaker Gordon Van Huizen offers some words of clarity in an effort to set the record straight. [Read More]
- Integration at the Edge
- Take the Enterprise Service Bus

JAVAPro Live! 2005
SEPTEMBER 11-13
Save the Dates!

Sign up online at: **www.javapro.com**

And while you're there, check out the complete FTPOnline network of technical sites for IT development professionals.

FTPOnline

Partners in Portal Federation Delivery

BEA and Tangosol combine to give enterprises WAN-capable, clustered session management and caching for BEA WebLogic Portal applications

Tangosol, a provider of data-grid and clustered caching solutions for large-scale Java and J2EE applications, is adding Wide Area Network (WAN) clustering, clustered workflow, and clustered caching to WebLogic Portal applications in an effort to federate large-scale, mission-critical applications through the new Web Services for Remote Portlets (WSRP) standard. This standard combines efforts from two Organization for the Advancement of Structured Information Standards (OASIS) technical committees—Web Services for Interactive Applications (WSIA) and WSRP—to simplify integration through a standard set of Web service interfaces for integrating applications and exploit new Web services when they are available. The combined WSRP and WSIA authorship of these interfaces provides reuse of presentation-oriented, interactive Web services and enables consuming applications to access a richer set of standardized Web services.

Tangosol Coherence enhances applications running on BEA WebLogic Portal through a blueprint for clustered and federated portal data that leverages WebLogic Portal's Custom Data Transfer extensions and Coherence's partitioned clustered caching. Using the Tangosol blueprint, portal applications can share large amounts of workflow and live data in real time across any number of federated portlets. This real-time sharing allows assembly of complex applications from an organization's reusable portlets.

Alex Toussaint, BEA product manager for WebLogic Portal said in a release statement that a lot of BEA customers want to take advantage of the opportunity in sharing workflow and live data across federated portals because, as they build complex portal applications, they see opportunities for reusing individual portlets that may be developed independently and may be deployed into separate application clusters and datacenters. Portlets can often be applicable to internal applications and to public-facing Web sites. Customers can use BEA WebLogic Portal with WSRP to federate them together and Tangosol Coherence to share live data and documents within the federated portals.

Specifically, Tangosol's Coherence*Web 3.1 is a module that now offers WAN-capable HTTP session clustering support for BEA WebLogic Portal applications. Coherence*Web provides HTTP session management for extreme-scale clusters that require very high levels of availability and scalable performance. By enabling the portal to cache portal and personalization data in clustered Coherence caches directly through the use of WebLogic Portal's native personalization (P13N) cache APIs, Tangosol is looking to intensify integration of Coherence into the WebLogic Portal.

Cameron Purdy, president of Tangosol, said in a release statement that the company will continue with its aim to offer enterprises high levels of availability and scalability for their applications, seamlessly and automatically, and that joint customers already are reaping the benefits of this Tangosol-BEA partnership. For more information visit BEA's Web site at www.bea.com, and visit Tangosol's Web site at www.tangosol.com.

Source: Tangosol Inc.

Developer Tools

JBuilder 2006

Borland Software's JBuilder 2006 is an upgrade to its Java IDE that includes new capabilities designed to help software teams collaborate more effectively in real time, across geographic boundaries, and with new, peer-to-peer, developer-collaboration features. It provides integrated application life-cycle support for requirements management, source code management, and unit testing. A new version of the Optimizeit application performance management toolkit is also now available as Borland's latest solution for isolating and resolving performance hazards during the development of J2EE applications. Both products are key components in Borland's application life-cycle management (ALM) solution, and the new versions are being integrated with other Borland ALM products. The needs of distributed teams were taken into account in the design of JBuilder 2006. New collaboration capabilities help individuals and teams work effectively with outsourced, offshore, remote, or distributed team members. Regardless of their geographic location, developers can collaborate to edit code, do visual design, and debug in real-time. Visit Borland's Web site for more information.

Borland Software
800-523-7070
http://support.borland.com

Vordel SOAPbox

The latest version of Vordel SOAPbox, a security testing tool for Web Services, lets you test Web services for security compliance by fast-tracking the creation and deployment of security frameworks to protect XML applications through internal and external security guidelines. Mark O'Neill, Vordel CTO, said Vordel's customers run SOAPbox as a client to test the security compliance of their Web services by generating security tokens, inserting them into SOAP messages, and firing them at secured Web services. Vordel SOAPbox 3.1 includes sample SOAP messages, XML encryption and decryption, DIME and MIME SOAP attachments, a graphical key store that loads multiple keys and certificates without using

command-line tools, the ability to remove security tokens from messages automatically, and the ability to specify transfer encoding. Visit Vordel's Web site to purchase SOAPbox or obtain a free trial copy.

Vordel
+353-1-603 1700
Fax: +353-1-603 1701
www.vordel.com/soapbox/

QCChart2D Charting, QCRTGraph

QCChart2D Charting and QCRTGraph Real-Time Graphics are object-oriented, 100% Pure Java software tools from Quinn-Curtis Inc. that enable developers to add real-time graphics and charting to Web pages and workstations using Java applets and applications. The products are designed for software professionals creating applications for financial markets, network monitoring, factory automation, process monitoring, R&D instruments, test systems, and SPC quality control. Both support linear, logarithmic, time/date and polar coordinate systems used by business, engineering, and scientific users. Data plot types include static and dynamic plotting of line plots, bar graphs, scatter plots, dials, gauges, meters, clocks, annunciators, and panel meter indicators. Both products are sold using a single-user developer license that entitles purchasers to free updates for two years. Visit the Quinn-Curtis Web site for more information.

Quinn-Curtis Inc.
508-359-6639
Fax: 508-359-4123
www.quinn-curtis.com

Reporting Tools

JasperAssistant 2.0

JasperAssistant 2.0 is a visual report designer for Infologic SA's JasperReports that makes it easy for you to create reports. JasperReports is an open source Java reporting tool that provides rich content on screen, to a printer, or into PDF, HTML, XLS, CSV, and XML file formats. Delivered as an Eclipse IDE plug-in, JasperAssistant runs under Windows and Unix/Linux and adds a WYSIWYG visual designer to the reporting engine. It provides an intuitive visual interface that lets you quickly design the layout of professional-quality reports. It has an integrated report compiler that is implemented as an Eclipse Builder, which allows automatic compiling of your report. The compiler's error messages link each error to a report object, making it easy to locate problems. You can preview your reports using live data from a JDBC database connection, an XML document, a JavaBeans array, or any custom data source. A single-user license is $129, and multiuser and site licenses are available. Visit Infologic SA's Web site to download a 21-day trial version or purchase the product.

Infologic SA
04-75-82-16-30
Fax: 04-75-42-18-33
www.jasperassistant.com

Database

DataDirect Connect Driver for JDBC

The DataDirect Connect for JDBC driver—from DataDirect Technologies, a unit of Progress Software Corporation—provides early access assistance for JDBC data connectivity to the Microsoft SQL Server 2005 database. Building on a history of SQL Server support, DataDirect Technologies offers in-depth migration guides and consultation for developers migrating to DataDirect's JDBC drivers on the Microsoft SQL Server 2005 database. The company's early access program allows customers to get an advance start on using and certifying their applications for SQL Server 2005 and enables developers to experience enterprise-class performance and scalability in the database. JDBC migration guides and limited telephone-based consultations for how-to approaches to migrating to the DataDirect Connect for JDBC drivers on SQL Server 2005 are also provided, regardless of the driver type in current use. DataDirect aims to simplify complex data connectivity issues for Java developers as more organizations migrate to SQL Server 2005, creating compatibility issues between versions of the database. Visit DataDirect's Web site for the early-access program's availability.

DataDirect Technologies
919-461-4200; 800-876-3101
Fax: 919-461-4526
www.datadirect.com

Compiled by Terrence O'Donnell

HOW TO REACH US

Editorial Offices: Fawcette Technical Publications, 2600 South El Camino Real, Suite 300, San Mateo, CA 94403; Phone: (650) 378-7100; Fax: (650) 570-6307; Web: www.javapro.com; e-mail: java-pro@fawcette.com.

Article Submission: To submit articles for publication please contact Terrence O'Donnell, Editor, todonnell@fawcette.com. Download the Author Guidelines at www.ftponline.com/javapro/code/12dec01/author_guide.zip.

Product Announcements: To submit announcements for new products or updates to existing products, please e-mail press releases to Terrence O'Donnell, Editor, todonnell@fawcette.com.

Reprints and Permissions: For all *Java Pro* editorial and advertising reprints contact Serv U Reprints, 101 Rhoades Way, Folsom, CA 95630; Phone: (916) 983-6562; Fax: (916) 983-6762.

To Quote from an Article: Please contact Susan LaCroix, 2600 South El Camino Real, Suite 300, San Mateo, CA 94403; Phone: (650) 833-7118; or e-mail: slacroix@fawcette.com. Specify the issue date and title of the article, the portion you would like to quote, and the purpose.

Photocopy Rights: Permission to photocopy for internal or personal use may be granted by Fawcette Technical Publications. Please contact Susan LaCroix at slacroix@fawcette.com for more information.

Customer Service and Subscription Information: For subscription orders, inquiries, or address changes please contact Customer Service, *Java Pro*, P.O. Box 3485, Northbrook, IL 60065-9819; Phone: (866) 387-5776; Fax: (847) 291-4816; for international inquiries call (847) 559-7309; or e-mail jva@omeda.com. Foreign and Canadian orders must be payable in U.S. dollars, plus postage. The surface rate to Canada and Mexico is $52.97 per year. For all other countries the air mail rate is $78.97 per year.

Back Issues: To order *Java Pro* back issues, call (650) 378-7100 or (800) 848-5523 and ask for Customer Service. Back issues cost $10. Additional postage will be charged to deliveries outside the USA.

Jini at Your Service

Discover how a revolutionary environment for distributed computing has evolved to be not unlike an SOA

by Alex **KRAPF**

Sun Microsystems introduced Jini originally as a technology that would allow devices to participate in reliable, distributed applications. Historically, it never found acceptance in that problem domain for several reasons: it requires a pretty large chunk of the Java run-time stack; it competes head-on with the UPnP specification that was supported by Microsoft in its dominant Windows platform; it's licensing was fairly nonstandard; and its continued support by Sun was not a certainty.

Go Online

Visit www.javapro.com for related resources. Simply type the Locator+ code into the field in the upper-right corner of the page.

Download

JP0511 Download all the code for this issue.

Discuss

JPTALK Discuss this article in the "Talk to the Editors of Java Pro Magazine" discussion forum.

Read More

JP0511AK_T Read this article online.

JP0411PV_T Read the related article "Achieve Optimal Performance" by Peter Varhol.

EA0403TM_T Read the related article "SOA: Debunking 3 Common Myths " by Tarak Modi.

JP0406PS_T Read the related article "Remote Access for Managed Applications" by Daniel F. Savarese.

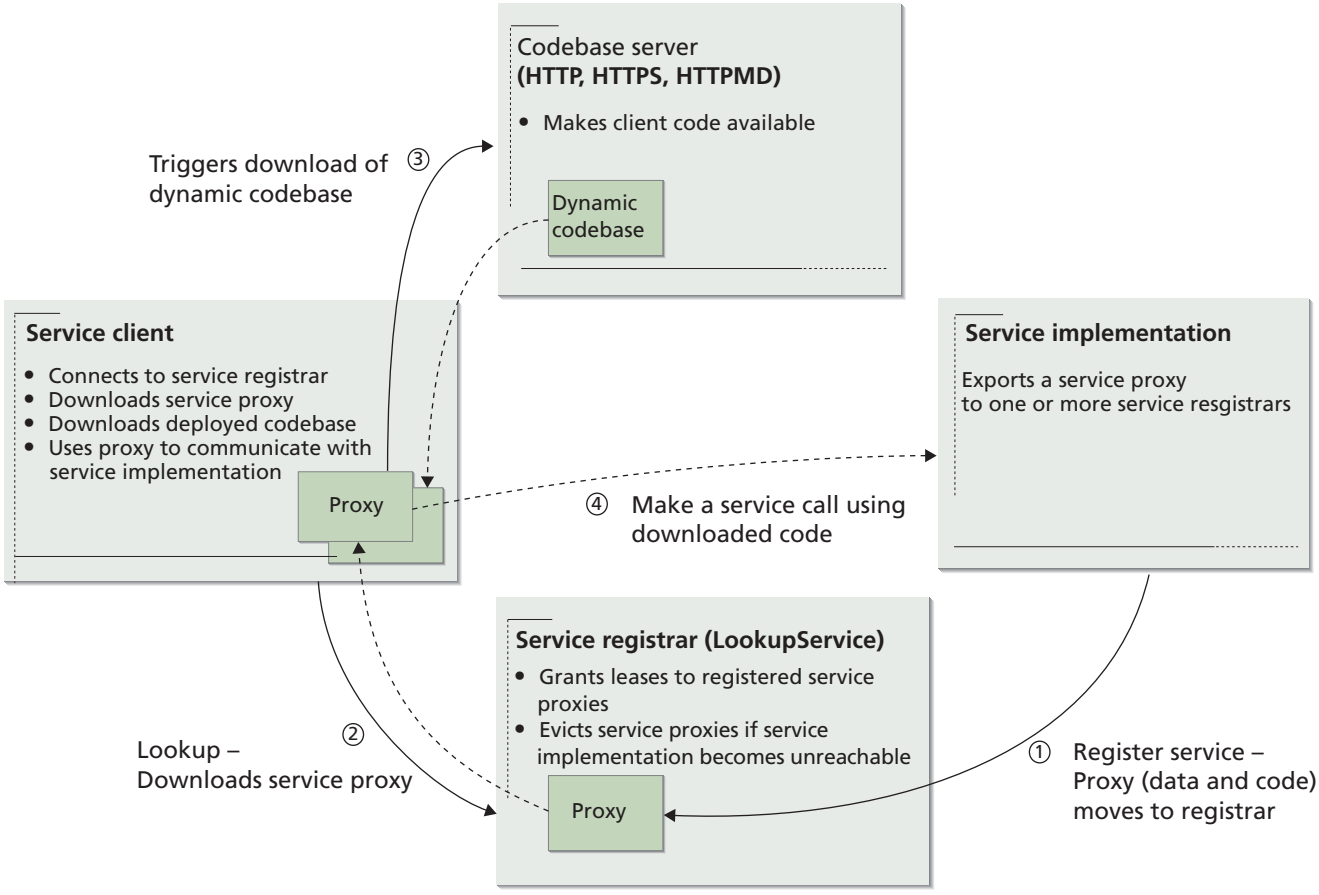

Figure 1 | The Whole Shebang In Jini's deployment and communications model the entire service discovery process consists of 1) registering the service, 2) downloading the service proxy, 3) downloading the dynamic codebase, and 4) making the client/server method call.

What most people didn't realize during the initial launch of Jini was that it also provided a gorgeous framework for building service-oriented applications. The failure to notice this important capability is easily understood: the term *service-oriented architecture (SOA)* had not even been invented yet. SOA was something that good client/server architects did instinctively or based on long, hard-won experience. Since those early days, a lot has happened.

Jini is now available in version 2.1 under the Apache 2.0 license. Sun and third parties have shown that Jini can work on small, embedded devices with a memory footprint of tens of kilobytes as well as on massively parallel super computers. Jini can be used from C/C++, .Net, or other clients and does not necessarily require a full Java stack.

How could Jini have been so misunderstood? Is it still relevant now that we have Web services and other technologies at our command? Let's take a closer look at Jini to answer these questions.

We shouldn't blame Sun all too much for Jini being misunderstood. After all, Java itself was originally aimed at applets and at powering small devices like cell phones or handhelds, and it took the majority of us a while to recognize its benefits for building powerful desktop and server applications. A revolutionary idea is by definition, well, revolutionary, and that means that its own creators frequently have no idea where it's going to take them. In the case of Jini, the technical vision was compelling: smart devices that just need to be plugged into a network or a bus or any communications channel, and they would make themselves known to their consumers; a standardized, simple lookup mechanism to identify a device that a consumer might wish to use; and a standardized, simple way to work with device and communication failures, acknowledging the fact that things break and network administrators occasionally make mistakes (no really, they do!).

Not So Fast

Well, it was not to be. Was a device vendor really going to add the necessary hardware to run the required Java Virtual Machine (JVM)? And even if they were to do that, would Microsoft add Jini support to its Windows operating system to take advantage of such devices? Of course not! The answer seemed so perfectly obvious to most people that they just looked at Jini as YADOAT (yet another dead-on-arrival technology).

However, not everybody discarded Jini right away, and—to Sun's credit—they kept developing and supporting it. Some early adopters looked at Jini and said, "Wow! Nothing in the specification says that I *have* to use it for devices. I could instead build applications that take advantage of the Jini features." These early adopters included hobbyists, visionaries, and places with *big* problems in the area of distributed computing: telcos, financial firms, and the military. To understand what they saw and what the others had missed, let's look at Jini's features. (See the sidebar, "The Jini Alternative" online at www.javapro.com for a summary of why Jini is a more attractive problem-solving alternative to other technologies.)

What is Jini? To take its definition straight from the Jini Architecture Specification (see Resources online at www.javapro.com for a link to the specification), Jini is a set of components that provides an infrastructure for federating services in a distributed system, a programming model that supports and encourages the production of reliable distributed services, and a set of services that can be made part of a federated Jini system and that offer functionality to any other member of the federation.

That definition sounds very software oriented, in fact, it sounds very much like a description of an SOA. Let's break down this definition. *Federating services* means that you take a group of independent, potentially distributed services and aggregate them into a single, dynamic distributed system. The difficulty here is that every single service can fail, and a federated service system has very nasty partial failure modes, which brings us to the second point: reliable distributed services. *Reliability* lies truly at the heart of a well-designed distributed system. Jini is arguably the first architecture that acknowledged the near certainty of partial system failure and turned it into a cornerstone of its design (see Resources online at www.javapro.com for more information on distributed computing).

The third part of Jini's definition in the specification is about a set of utility services that offer *functionality* to other services. These services include such necessities as a transaction service and a persistence service. The best way to understand the core ideas behind Jini is to give yourself a traditional problem, and then describe what your application might look like if you solved the problem using Jini.

Put It to the Test

Consider a problem example. Let's assume you have been charged with developing a client/server API that allows your customers to submit electronic orders. The system must scale in both number of customers and number of orders' dimensions. Customers must not have to install frequent updates, and they must be able to use different versions of the software concurrently. There must not be any system downtime because of internal problems (luckily for you, someone else has to worry about the Internet- and network-related parts of this problem). System degradation under load must be gradual, and so forth.

You probably get the idea: we're talking about all the hard things like automatic remote deployment, failover, load balancing, automated failure detection, and so on. Let's look at the service API that we want to present to our customers (we're definitely looking at a 0.1 version here):

```
INTERFACE ORDERSERVICE
{
  PUBLIC RESPONSE SUBMIT (
    ORDER O ) THROWS
    REMOTEEXCEPTION;
  PUBLIC RESPONSE VALIDATE (
    ORDER O ) THROWS
    REMOTEEXCEPTION;
  PUBLIC RESPONSE CANCEL (
    ORDER O ) THROWS
    REMOTEEXCEPTION;
}
```

Order is an interface that gives access to all the required input data, and Response is an interface that can be used to query the required output data. Using interfaces instead of classes for the data wrapper types will give us important benefits for deployment and maintenance, but we'll get to that later. The methods throw a RemoteException if anything goes wrong at the framework/communications level. Our users will write applications such as this service client:

```
TRY
{
ORDERSERVICE SERVICE =
  CONNECTTOSERVICE(
  USER, PASSWORD );
ORDER THEORDER =
  ASSEMBLEDATAFROMGUI();

IF( SERVICE.VALIDATE(
  THEORDER ).ISOK() )
  SERVICE.SUBMIT( THEORDER );
}
CATCH ( REMOTEEXCEPTION RE )
{
...
}
```

There is hardly anything revolutionary here. One thing you might notice is that we're not using any transactions. Let's pretend that we're carrying around Transaction objects in the API as well. They are available easily enough in Jini, but I want to focus on the Jini-specific concepts and not on transactions.

One thing to think about is the number of remote calls you see here: there must be at least one, maybe more than one, in connectToService(), and then there's one for validate() and one for submit(). We'll revisit these numbers shortly.

Notice also the utility method connectToService(), which gives us an instance of a type that implements our OrderService interface. We'll have to implement it in terms of our distributed programming API of choice. Let's take a look at a typical Jini implementation of the connectToService() method, and then we'll look into some details of what's happening under the hood and how it is different from most other distributed architectures. Here is a client connecting to a service:

```
ORDERSERVICE CONNECTTOSERVICE(
  STRING USERNAME,
  STRING PASSWORD )
  THROWS REMOTEEXCEPTION
{
  LOOKUPLOCATOR LOCATOR =
   NEW LOOKUPLOCATOR(
   "JINI://VENDOR.COM" );
  SERVICEREGISTRAR REGISTRAR =
   LOCATOR.GETREGISTRAR();
  SERVICETEMPLATE TEMPLATE =
   NEW SERVICETEMPLATE( NULL,
   NEW CLASS[] {
   ORDERSERVICE.CLASS }, NULL );

  RETURN (ORDERSERVICE)
   REGISTRAR.LOOKUP( TEMPLATE );
}
```

We're of course ignoring the username and password arguments here, but it would be easy to use additional argument versions of the APIs and the Java security API to do the same thing with your choice of authentication protocol.

Make a Discovery

If you've never looked at Jini in the past, you will probably assume that you have a pretty good idea of what's going on under the hood in this code. The LookupLocator and ServiceRegistrar classes are something like a CORBA NamingService. It has the unique identifier of an object and gets back

a proxy to a server implementation of the service interface, probably something like an OrderServiceClientImpl object that was generated by a Jini tool based on the declared service interface.

Somehow I just pass an interface type to the lookup method, which probably means that it does some classname-based lookup or some such thing. Those are some excellent assumptions; however, fortunately they're wrong. If you take a closer look at the lookup and discovery process first, you can see we are using a unicast discovery, which is a discovery mechanism that uses a known URL to locate a lookup service. The lookup service is essentially your factory for all other services that you might be interested in.

If your network configuration supported it, we could also have used a multicast discovery mechanism, removing the need for a known URL. What does Jini do for us here? In the case of multicast service discovery, you have no single point of failure in your lookup process. If just one of a hundred lookup service instances is reachable, you will discover it. In the case of unicast discovery, you can theoretically still avoid single point of failure by having a network layer that can reconfigure or reroute dynamically to a reachable host.

This feature is one of the benefits of Jini; it's easy to create redundant systems. Once you have discovered your lookup service instance, you look for an instance that implements your OrderService interface. As it turns out, we just stated exactly how lookup works in Jini. You don't look for services based on names or identifiers, you look for them based on their type (the arguments that are null in the example allow you to specify additional constraints). One of Jini's most crucial concepts is that all you really have to know about a service that you wish to consume is the service's interface type.

What if you wanted to acquire a service instance that implements both OrderService and PaymentService interfaces? You simply ask for a service instance that implements both interfaces. Notice that the second argument to the lookup template constructor is an array of types. You simply pass an array containing both interface types, and you will get back null if there is no such service, or a valid service proxy object if there is.

Incidentally, even if you asked for the OrderService only, you might still get an instance that implements more than one interface. You will probably not be able to use the other implemented interfaces, and you definitely shouldn't rely on their presence because someone might deploy another OrderService instance (this time without PaymentService), and then you might discover that instance rather than the combined instance.

Make the Call

Now that we have our client-side service proxy instance, how does the server call work? Does Jini use Java Remote Method Invocation (RMI), IIOP, or a proprietary protocol? The answer is, all and none of the above. Jini is totally protocol independent. The service proxy that you are working with in your client application is downloaded from the lookup service and a codebase server. That's right: you don't have to have the service client code deployed at all! The fact that the service implementation is downloaded has several important consequences.

> We shouldn't blame Sun all too much for Jini being misunderstood. After all, Java itself was originally aimed at applets and at powering small devices like cell phones or handhelds

First, obviously the service proxy does not have to be deployed on the client for the application to work. The only types that need to be deployed on the client are the service interfaces and interfaces that they depend on (like Order and Response). This aspect has a huge impact on ease of patching and deploying. Basically, you can fix bugs and make implementation changes in your API without ever stopping clients or servers. Simply update the codebase server from which your client downloads the service proxy, and newly connecting clients will get the fixed code. Clients already running will get the new codebase after they reconnect to an updated service (you need to take some care in your client code to allow that to happen). By making sure that you only extend the interface rather than modifying it, you can concurrently support different client versions as well.

Second, the downloaded service proxy decides which protocol is used for client/server communications; one of the easiest protocols to use is RMI, in which case you have virtually no additional work to get the client and server talking with each other. However, you might even decide not to use any communications protocol.

What, no protocol at all? Yes, that's how you do a smart client in Jini. Not every service method has to result in a remote call. As a service implementer, you have the choice, which is extremely beneficial if we consider the validate() method in our service. It is considered self-evident that client-side validation is something you cannot rely on in your server, but it can be crucial for application performance to perform validation on the client rather than through a remote call. It is thus a constant challenge to make sure that a client uses the same prevalidation code that the server uses, and it can cause troublesome bugs when the two validation schemes get out of sync. In the Jini world, your client proxy implementation would be the download code that is also used by your server. You could perform the prevalidation on the client without a remote call.

Third, the same service may be registered with more than one lookup service. Any one lookup service can deliver the service proxy to the client. Multiple instances of one service can be registered with multiple instances of a lookup service. You will never know which one you acquire, but the lookup services typically use a simple round-robin scheme to distribute load between multiple available instances.

You're absolutely right if you are nervous about having dynamically downloaded code executing on your client. Rest assured that Jini provides simple ways to authenticate all participating parties and to ensure the integrity of the downloaded code through standard security mechanisms like JAAS, SSL, or Kerberos. Figure

1 provides an illustration of the entire service discovery process, all the way to a client/server method call.

Trivial Failure Recovery

Let's say you have a complex application that uses 20 different federated services located on 10 different hosts. Now all of a sudden one of the server hosts dies and takes three services down with it. In traditional frameworks, it can be pretty hard for an application to recover from such a failure. In Jini, it can be trivial.

> Jini is arguably the first architecture that acknowledged the near certainty of partial system failure and turned it into a cornerstone of its design

Assume for a moment that no service existed as a singleton, that is, that every service has multiple instances running on different hosts. Each of these service instances is registered with all lookup services that themselves are running on different hosts.

Here's a typical failure scenario: a client holds a proxy to one of the dead services and wants to call a remote method on the proxy. This call will eventually result in a RemoteException because of communication failure or timeout. The client doesn't just throw up its hands in exasperation; rather, it decides to look up the service again. It will receive a new service proxy to one of the other service instances that are still up and running. Now the application can continue with the new service proxy as if nothing had happened.

How does Jini know that the service is down? How do we make sure that the lookup service doesn't hand out the same proxy again? The key to Jini's responsiveness lies in the fact that a service is pretty much expected to fail or become unreachable at some point. To account for this event, a service implementation cannot register itself with a lookup service for an infinite amount of time.

When a service implementation registers itself with the lookup service, it is granted a lease—a time-limited registration. If the service wants to remain available as an active service, it has to renew the lease from time to time. If it cannot do that because its host has crashed or has become unreachable on the network, the service is evicted by the lookup service and cannot be discovered anymore by clients. The lease interval is configurable and can play a big role in how responsive a system is to failures.

Jini provides a number of utility classes that do all the heavy lifting for lease renewal and management, so it's very easy to write a server application (or, in other words, a service implementation) that scales well and is reliable by simply running it on multiple hosts. Jini manages failure as part of the framework. This code illustrates how simple it can be to create a server registering itself with Jini (features like configuration and security are left out to keep it simple):

```
PUBLIC VOID REGISTERSERVER(
  SERVICEREGISTRAR REGISTRAR )
{
  LEASERENEWALMANAGER LRMGR =
    NEW LEASERENEWALMANAGER();
  SERVICEITEM ITEM =
    NEW SERVICEITEM( NULL,
    NEW ORDERSERVICEIMPL(),
    NULL );

  // HERE WE MAKE THE SERVICE
  // AVAILABLE FOR LOOKUP
  // WE ASK FOR INFINITE
  // REGISTRATION, WHICH WE
  // PROBABLY WON'T BE GRANTED,
  // SO WE'LL HAVE TO MAKE SURE
  // TO RENEW LATER ON
  SERVICEREGISTRATION SREG =
    REGISTRAR.REGISTER( ITEM,
    LEASE.FOREVER );

  LRMGR.RENEWUNTIL(
    REG.GETLEASE(),
    LEASE.FOREVER, THIS );
}
```

In this snippet we ask to keep the service registered forever, but the lookup service will probably not grant that request. Instead, it will return a lease that contains the eviction time for our service. To stay registered forever, our service will have to renew that lease before it expires—for example, by calling renewUntil().

Eviction Notice

What could you achieve by choosing different values for the requested lease time? You might want to give a service a lease of no more than 30 seconds, meaning that 30 seconds after registration the lookup service will automatically evict our server, and clients will not be able to find it anymore. You might want to evict the server to check system load or aggregated service load, for example, every 30 seconds and decide that you don't need 25 instances of this service running right now. In that case you would simply not reregister the evicted service; if on the other hand you decide that you want to keep the service alive, you simply reregister it again. Your choice of lease interval can determine how dynamic a distributed Jini system reacts to changes in its hosting environment.

People have built frameworks that support the dynamic migration of services on top of Jini. Borrowing a term from the J2EE world, they are typically called *service containers*, and they provide service deployment and management capabilities that go beyond the built-in Jini features.

Jini is a technology whose time has come. Jini and its add-on projects allow you to quickly and inexpensively build secure, reliable, resilient, distributed applications. It is very much ready for prime time, as proven by its use at places like Orbitz, large U.S. government agencies, and the financial industry. However, you don't have to be a huge enterprise to benefit from Jini. If all you need is dynamic client updating; just leverage Jini's code mobility feature. If the Jini code mobility feature makes you nervous, just use Jini's service features with local codebases. Jini is completely at your service! JP

Alexander Krapf is president and cofounder of CodeMesh Inc. Alexander has over 15 years of experience in software engineering, product development, and project management. He has also worked for IBM, Thomson Financial Services, Hitachi, Veeder-Root, and Document Directions Inc. Contact Alexander at **alex@codemesh.com**.

Migrate J2EE Applications for EJB 3.0

Avoid complexities when migrating applications to use the new standard in server-side business logic programming

by Debu **PANDA**

The programming model for Enterprise JavaBeans (EJBs) has been simplified dramatically in EJB 3.0 and is being hailed by Java developers as the new standard of server-side business logic programming. Meanwhile, the existence of thousands of J2EE applications written with earlier versions of the EJB API has raised concerns about both the interoperability of EJB 3.0 with these applications and migrating the applications to use EJB 3.0.

Major application server vendors that already provide EJB 3.0 features will continue to support EJB 2.x in the new EJB 3.0 container. This support means that applications written with EJB 2.x will continue to run without any change whatsoever. However, some organizations will certainly want to migrate their applications to use the EJB 3.0 API to improve maintainability by simplifying their code and take advantage of the performance and scalability benefits of the persistence API.

Let's look at some of the issues relevant to migrating applications to EJB 3.0. The topics discussed here compose a subset of all possible issues a migration effort will encounter, and because the spec is not finalized other issues may still arise.

The main goals of EJB 3.0 are to simplify the programming model and to define a persistence API for the Java platform. Some general changes to the EJB spec that will simplify life for EJB developers are: EJB artifacts are Plain Old Java Objects (POJOs), XML descriptors are no longer necessary, annotations may be used instead, defaults are assumed whenever possible, unnecessary artifacts and life-cycle methods are optional, and the client view is simplified by dependency injection.

Standardization of the POJO persistence model (such as the one used by Oracle TopLink and JBoss Hibernate) within the context of J2EE finally provides users with the inheritance and polymorphism that have until now been available only outside the container or in proprietary products. Furthermore, the POJO entity beans can now be used and tested both inside and outside the EJB container.

Migrate Session Beans

The main changes in EJB 3.0 session beans simplify development by making beans POJOs that use annotations instead of XML descriptors and dependency injection instead of complex JNDI lookup. Hence, it is very easy to migrate the session beans to use the EJB 3.0 programming model. The changes in EJB 3.0 sessions fall into two categories: the server side and the client side. By client side we mean using resources, EJBs, and so on from other EJBs. Using EJBs from other types of clients outside the EJB container will be standardized with the J2EE 5.0 specification.

Some application servers allow migration of server-side components without affecting the clients, thus allowing incremental migration of applications.

There are some changes that need to be made to EJB 2.x session beans to migrate to EJB 3.0. In EJB 3.0 the remote and local interfaces do not have to implement javax.ejb.EJBObject or javax.ejb.EJBLocalObject. The component interface can become a business interface, and @Remote annotations can be used to mark remote interfaces.

Go Online

Visit www.javapro.com for related resources. Simply type the Locator+ code into the field in the upper-right corner of the page.

Download

JP0511 Download all the code for this issue.

Discuss

JPTALK Discuss this article in the "Talk to the Editors of *Java Pro* Magazine" discussion forum.

Read More

JP0511DP_T Read this article online.

JP050622MS_T Read the related article "Prepare for New EJB 3.0 Peristence" by Merrick Schincariol and Doug Clarke.

JP041222DC_T Read the related article "Select the Best Persistence Architecture" by Doug Clarke.

JP041208WK_T Read the related article "Using JDO for Transparent Persistence" by William Korb.

Let's look at two examples of the same EJB 2.x interface, before and after migration to EJB 3.0. RemoteExceptions are no longer thrown by the methods on the EJB 2.x remote interface:

```
import javax.ejb.EJBObject;
import java.rmi.RemoteException;

public interface HelloWorld
  extends EJBObject {
  void sayHello(String name)
    throws RemoteException;
}
```

and the EJB 3.0 remote interface:

```
import javax.ejb.Remote;

@Remote
public interface HelloWorld {
  void sayHello(String name);
}
```

Bean classes do not need to implement javax.ejb.SessionBean—rather, their business interfaces. Life-cycle methods that are not required do not need to be present on the bean class, and callbacks may be indicated by annotations. Here is the EJB 2.x stateless session bean:

```
import javax.ejb.SessionBean;
import javax.ejb.SessionContext;

public class HelloWorldBean
  implements SessionBean {

  public void ejbCreate() {}
  public void ejbActivate() {}
  public void ejbPassivate() {}
  public void ejbRemove() {}
  public void setSessionContext(
    SessionContext ctx) {}

  public void sayHello(
    String name) {
  System.out.println(
    "Hello " + name);
  }
}
```

and here is the stateless session bean migrated to EJB 3.0:

```
import javax.ejb.Stateless;

@Stateless
public class HelloWorldBean
  implements HelloWorld {
  public void sayHello(
    String name)  {
    System.out.println(
      "Hello " + name);
  }
}
```

Stateful session bean ejbCreate() methods become business methods used at initialization time. Removal methods are annotated with @Remove. Optionally, you can migrate session bean code to use annotations for transactions and security.

EJB 3.0 simplifies the use of resources and EJB references by making use of the principle of dependency injection. Injection of resources or references can occur through either annotation of the injection target or specification of the target in the ejb-jar.xml descriptor. Table 1 lists some of the differences.

For example, if you are using the CartEJB in another EJB named OrderBean, with EJB 2.x you have to make a reference to the CartEJB in the deployment descriptor using ejb-ref:

```
<ejb-ref-name>MyCart
</ejb-ref-name>
  <ejb-ref-type>Session
  </ejb-ref-type>
  <home>CartHome</home>
  <remote>Cart</remote>
```

Do a lookup of the home interface for the CartEJB, using JNDI, and create an instance of the CartEJB:

```
Object homeObject =
context.lookup(
  "java:comp/env/MyCart");
CartHome home = (
  CartHome)PortableRemoteObject.
  narrow(homeObject,
  CartHome.class);
  Cart cart =(
    Cart)PortableRemoteObject.
    narrow(home.create(),
    Cart.class);
cart.addItem("Item1");
```

After migration to the EJB 3.0 injection pattern, this code can become much simpler and would look like this:

```
@EJB Cart cart;
  public void addItems() {
    cart.addItem("Item1");
    }
```

A session bean can also be used as a façade for CMP entity beans, giving rise to a migration strategy for moving CMP entity beans to the EJB 3.0 persistence API.

	EJB 2.x	EJB 3.0
Using another EJB:	ejb-ref in XML descriptor Do JNDI lookup to get the home interface Call home.create to obtain instance	The ejb-ref no longer requires the home interface. Change the home interface to the business interface. Remove home.create and directly invoke the methods on the EJB. Optionally annotate the EJB business interface property/field to obtain an instance.
Resource access:	Do JNDI lookup to obtain a resource	Optionally annotate a resource property/field to obtain a resource.

Table 1 | Inject Simplification EJB 3.0's principle of dependency injection simplifies the use of resources and EJB references. Annotation of the injection target or specification of the target in the ejb-jar.xml descriptor provides for the occurrence of resources or references. Compare the differences.

More Migrations

Migrating Message-Driven Beans (MDBs) to use EJB 3.0 is by far the easiest migration task. In EJB 3.0, MDBs do not have to implement the javax.ejb.MessageDriven interface but can instead be annotated with @MessageDriven. Resources and EJB references can be injected into MDBs in the same way as session beans. Similarly, you can inject the context (MessageDrivenContext for MDB) into the message-driven bean. Table 2 summarizes the changes in MDB between EJB 2.x and EJB 3.0.

Persistence is one of the greatest challenges facing Java developers, compounded further

by the lack of a standard persistence API for the J2EE platform. J2EE applications typically use one of these persistence choices: EJB 2.x CMP entity beans; a POJO persistence framework such as Oracle TopLink, JBoss Hibernate, or another custom O/R mapping framework; Data Access Objects with JDBC; and Java Data Objects (JDO).

Of these options, migrating EJB 2.x CMP entity beans presents the most interesting case because the change in programming model brings the EJB 3.0 persistence API to the same level of "POJOness" as other persistence solutions, which is worth taking some time to explore.

EJB 2.x CMP entity beans represent coarse-grained data objects and follow the distributed component model that includes security, transaction, and concurrency management. Because of these facilities they also carry the container management costs associated with it. EJB 3.0 entities are lightweight objects that can model fine-grained data. The difference between the 2.x and 3.0 models extends beyond simply converting the classes but should be considered at the modeling domain level. This difference does not mean that they cannot be converted, but that they should be migrated with some forethought and consideration of how they might best be remodeled.

Let's take a detailed look at some of the issues you will encounter as you migrate an EJB 2.x CMP entity to use the new EJB persistence API.

Remodel your entities. With EJB 3.0 entities becoming POJOs, it is probably the best idea to remodel your entities and leverage richer OO modeling benefits such as inheritance, polymorphism, and so on. The details of remodeling of entity beans vary from one application to another and are beyond the scope of this article.

Data transfer objects. The Data Transfer Object (DTO) pattern is a common pattern that allows data from entities to be accessible directly to nonlocal clients outside a transaction. It involved creating simple container objects to hold the entity data and transferring them to any tier as necessary. The EJB 3.0 specification makes the DTO pattern unnecessary because entities can already be shipped wherever they are required. The EntityManager API that is used for CRUD operations for entities allows detachment and merging of detached objects. An object can be detached from persistence storage and can be sent to the client, and then the changes can be merged/synchronized using the EntityManager.merge() method after the client sends it back to the server making updates to the object locally.

If DTOs already exist in an application, it is quite likely that they are the shortest path to making the EJB 2.x entities POJOs. The DTO becomes the candidate for the new entity bean if you simply add the logic into it and annotate/map it accordingly.

> The main goals of EJB 3.0 are to simplify the programming model and to define a persistence API for the Java platform

More Issues

Migration of component interfaces. Business interfaces are optional for EJB 3.0 entity beans, and if they exist, they are only regular Java interfaces (POJIs). You can either convert existing local or remote component interfaces to regular Java interfaces by no longer extending the javax.ejb.EJBObject or EJBLocalObject interfaces, or discard them entirely.

EJB 2.x	EJB 3.0
Implements javax.ejb.MessageDriven	POJO: May annotate with @MessageDriven
Specify destination type, name, and so on in the deployment descriptor.	May be annotated with @ActivationConfigProperty
MessageDrivenContext is acquired using setMessageDrivenContext().	MessageDrivenContext is achieved using dependency injection—for example: @Inject javax.ejb.MessageDrivenContext mc;
Resource usage, such as Queue or Topic, is a required resource-ref in the deployment descriptor and JNDI lookup.	Can be used employing dependency injection: @Inject private Queue destQueue;

Table 2 | MDB Migration MDBs do not have to implement the javax.ejb.MessageDriven interface in EJB 3.0, but they can instead be annotated with @MessageDriven.

Conversion to POJO. EJB 3.0 entity beans no longer expect that the bean class implements the javax.ejb.EntityBean interface. They may extend any class and implement virtually any interface. The EntityBean interface can be removed from the implements clause. You have to convert the entity class and the get/set access methods from abstract to concrete. The previously virtual container-managed fields (cmp-fields) should have concrete implementations added to them.

EntityContext. Lightweight entities now inherit their context from the calling method and have no context of their own. You must remove the EntityContext and replace use of the context by any methods with suitable alternatives. For example, the getPrimaryKey() method would be replaced with a method on the bean or business interface that exposes the primary key to the user.

Home interfaces. The typed local and remote home interfaces in EJB 2.x are replaced by functionality in the EntityManager. Local and remote home interfaces should no longer be used, and clients that look them up should be converted to call the EntityManager directly to obtain or access the bean.

There are four different types of methods in the home interface: create methods, remove methods, home methods, and finders. The create methods can be converted easily into constructors that initialize the state information on the instance. The remove methods should be redirected to the EntityManager. The home methods may remain as they are because they can be invoked on any instance, regardless of the state of the instance. I'll discuss Finder methods later.

Optionally, you can keep the local home interface that clients use as a helper class for finders and factory style creation to minimize client code change and produce a nice clean usage pattern for your entity bean.

Migration of life-cycle methods. EJB 2.x has a strict requirement to implement the EntityBean interface that defined the complete suite of life-cycle methods. Some of these life-cycle events no longer apply. Table 3 sug-

EJB 2.x life-cycle method	What to do in EJB 3.0
ejbCreate()	Implement logic in init methods/constructors.
ejbPostCreate()	Implement logic in a constructor, or create a business method and annotate it as @PostPersist.
ejbRemove()	Create a business method, and annotate it with @PreRemove.
setEntityContext() unSetEntity-Context()	No longer applies.
ejbActivate()	Create a business method, and annotate it as @PostLoad
ejbPassivate()	No longer applies (it can be removed).
ejbStore()	Create a business method, and annotate it with either @ PrePersist or @PreUpdate.

Table 3 | Migration Strategies Because some of the life-cycle events in EJB 2.x no longer apply, here are some possible strategies for migrating existing implementations of these life-cycle methods.

gests strategies for migrating existing implementations of these methods.

Migration of O/R mapping. Before EJB 3.0, O/R mappings were always in the domain of the vendor and, as such, were typically stored in a vendor-specific deployment descriptor. Now that the O/R mappings have been integrated into the EJB standard, they can be defined either as annotations on the bean class or in a standard XML file. Most vendors should be providing ways to translate their mappings into the standard EJB mappings, either through a migration tool or from their graphical mapping editors.

Exception handling. Exceptions are in a different Java package and are not checked; the same CreateException, RemoveException, and FinderException catch phrases do not apply.

Even More Issues

Migration of finder methods. In EJB 2.x, the finder methods were defined in a home interface, and the EJB QL query for the finder was specified in the deployment descriptor. There are a few options for migrating EJB 2.x entity bean finders to EJB 3.0. The finder method can become a method in any class, and the EJB QL can be incorporated into either a named query or a dynamic query. Here is a finder method example defined in EJB 2.x in the deployment descriptor:

```
<query>
  <query-method>
    <method-name>findAllByName
    </method-name>
    <method-params>
      <method-param>String
      </method-param>
    </method-params>
  </query-method>
  <ejb-ql>
    SELECT OBJECT(c) FROM
    Customer c WHERE c.name LIKE
    ?1
  </ejb-ql>
</query>
```

A named query would look like this:

```
public Collection findAllByName(
  String nameString) {
  return getEntityManager()
  .createNamedQuery(
    "findAllByName")
    .setParameter(
      "custName", nameString)
    .getResultList();
}
```

This example assumes that a NamedQuery annotation is defined this way:

```
@NamedQuery(
  name="findAllByName",
    queryString=
    "SELECT OBJECT(c) FROM
    Customer c WHERE c.name
    LIKE :custName")
```

Container-managed relationships. In EJB 2.x, the container was responsible for updating the other side of a relationship when one side was updated. EJB 3.0 opted to require the bean to manage its own relationships (to allow out-of-container entity testing and the like). Therefore, you must add management code to beans that have existing container-managed relationships, which is typically a one-line code change:

```
public addOrder(Order order) {
  getOrders().add(order);
}
```

Adding the management code would require the backpointer assignment:

```
public addOrder(Order order) {
  getOrders().add(order);
  order.setCustomer(this);
}
```

Migrating CMP clients. In EJB 2.x, local and/or remote clients accessed entity beans. A well-documented best practice was to keep the entity beans local and wrap them in a session façade. In EJB 3.0, entities, being regular Java objects, are always local and can never be remote (RMI) objects. They are obtained and queried for through the EntityManager API. Client code (a session bean in the case of a session façade) must change its use of entity home or component-specific methods to use the EntityManager to access the entities on which it operates.

Local and remote entity references used to be required declarations in the deployment descriptor. In the client code the home would be looked up in JNDI, and home operations such as creates and finds could then be performed. Here's an example of a client lookup:

```
public void createNewCustomer(
  String name, String city)
throws CreateException,
  NamingException,
  RemoteException {
InitialContext ctx = new
  InitialContext();
CustomerHome home =
  (CustomerHome) ctx.lookup(
  "java:comp/env/ejb/CustomerHome"
  );
CustomerLocal customer = null;
  customer = home.create(
  name, city);
}
```

The migrated version of this code would look like this:

```
@Resource private EntityManager
  em;
public void createNewCustomer(
  String name, String city) {
```

```
    Customer cust = new
    Customer();
    cust.setName(name);
    cust.setCity(city);
    em.persist(cust);
}
```

It is clear from the preceding discussion that migrating EJB 2.x entity beans to EJB 3.0 is the most complex task, will have an impact on clients, and needs careful planning.

Migrating POJO applications. Some frameworks have been persisting Java objects to relational databases for a long time, and vast numbers of applications are written by use of O/R frameworks. Applications that are currently using a POJO persistence framework are well positioned to migrate to the EJB persistence API because the domain objects are already Java objects. Further, the persistence frameworks' transaction mechanisms and session-level APIs are similar to those of the new EJB persistence API.

The result is that very little code rework is required. If you are currently evaluating a persistence framework for your J2EE applications, probably using a POJO persistence framework is the best choice because it can easily get you to the EJB 3.0 Persistence API.

Get to It

Change the session APIs used in the persistence framework to EntityManager APIs. Change proprietary O/R XML to O/R mapping annotations or O/R XML defined by the EJB 3.0 persistence API.

The technical details of migration of POJO persistence to EJB 3.0 are covered in the article, "Prepare for the New EJB 3.0 Persistence API" (FTPOnline, June 2005)—see Resources online at www.javapro.com.

Migration of J2EE applications to EJB 3.0 is certainly not rocket science; however, like any other migration effort, it takes education, resources, and planning. The first step is to become aware of where the platform is moving and understand where it plans to end up. Once you understand the technology, undertake planning to decide the best strategy for getting there.

The prudent approach to migration is to migrate a subsection of the application before applying the practices to the whole. This strategy will ferret out many of the difficult migration issues and allow for experimentation to discover which approach might be best suited to the application. Features such as interoperability between EJB 2.x and EJB 3.0 components are critical when you're pursuing this kind of strategy. You can start trying out EJB 3.0 with an early implementation, and get ready for migrating to EJB 3.0. JP

Debu Panda is a principal product manager of the Oracle Application Server development team and is responsible for the EJB container and Transaction Manager. His J2EE-focused blog can be found at http://radio.weblogs.com/0135826/. Contact Debu at **debabrata.panda@oracle.com**.

Investigate JDBC Problems

Take a look at several helpful troubleshooting practices for JDBC connection pool issues

by Maria **SALZBERGER**

Configuration of a JDBC connection pool or using nonrecommended programming techniques can lead to many different issues around the JDBC pool connections, the related databases, or the WebLogic Server instance. Also, the underlying database, the network configuration, and architecture can lead to issues with JDBC connections.

Let's look at troubleshooting tips for several of these issues and information on how to investigate JDBC problems further. Note that not all of the procedures here would need to be done. Carrying out selected procedures can solve some issues.

Let's begin with JDBC connection pool issues—ResourceExceptions from the JDBCConnectionPool. In certain situations, weblogic.common.ResourceException no resources available error messages are thrown during JDBCDataSource.getConnection() calls. A ResourceException is thrown when a getConnection() request through a DataSource to a JDBC connection pool can not be satisfied because either no connection in the pool or no thread is available to handle the connection request. The cause of the missing resource could be either a connection leak in the application, too few threads or too few connections (MaxCapacity), or connection peaks.

Connections used from the JDBC pool need to be closed after being used by the application code. If close() is not called, connections are not freed and not available for reuse. A possible symptom for a connection leak for Oracle JDBC connection pools can be this error message:

```
ORA-00020 - maximum number of
  processes (600) exceeded
```

WebLogic Server provides a connection leak-detection feature. If you suspect that the application does not close all JDBC pool connections correctly, enabling this feature can help you analyze the situation. This property can be set per connection pool. Documentation on the ConnLeakProfilingEnabled is available at BEA's dev2dev Web site (see Resources online at www.javapro.com). This property can be set through the WebLogic Server administration console. Note that this feature should not be used on production systems as it may have a performance impact.

Methodologies for how to investigate connection leaks will be handled in a support pattern that will give detailed information about how to analyze and resolve this problem. As soon as it is available, it will be posted to the BEA Support Patterns page (see Resources online at www.javapro.com).

Connection Peaks

JDBC connection pools must be configured to allow enough concurrently active database connections as needed by the application. A general rule of thumb here is to configure the number of database connections in the JDBC pool (MaxCapacity) to be equal to the number of execute threads. This value then limits the amount of database connections that can be active (enlisted in a transaction) in parallel, so capacity planning needs to consider this factor.

There may be situations where you see lots of messages in your WebLogic Server log file that state

Go Online

Visit www.javapro.com for related resources. Simply type the Locator+ code into the field in the upper-right corner of the page.

Download

JP0511 Download all the code for this issue.

Discuss

JPTALK Discuss this article in the "Talk to the Editors of *Java Pro* Magazine" discussion forum.

Read More

JP0511MS_T Read this article online.

JP041222DC_T Read the related article "Select the Best Persistence Architecture" by Doug Clarke.

JP0406KJ_T Read the related article "All that JAAS" by Kevin Jones.

JPEP030725CP_T Read the related article "JDBC, CMP, or JDO?" by Dr. Wilson Cheng and Pinaki Poddar.

that new connections to the database have been created for one of your JDBC connection pools. There may be situations where connection peaks occur, and during the same time the application gets ResourceExceptions, which implies that no connections are available in the pool.

The simplest reason for this situation is that the load for your application has peak times so that lots of connections are needed at the same point in time. However, if an analysis of the application shows that more connections are opened and reserved than are used by the application code, and connection leaks are excluded, most probably the refresh feature for the JDBC connection pool is enabled and the interval is set too frequently. To turn off refresh functionality: set TestFrequencySeconds to 0 in WebLogic Server 8.1 and later, set RefreshMinutes to 0 in WebLogic Server 7.0, and set RefreshMinutes to 9999999 (the maximum value here is 35791394) for prior versions. A setting of 0 will fall back to the default value of 5 minutes.

JDBC connection pool refresh functionality is intended to test all currently unused connections in the pool using the test table. It will refresh connections if needed, that is, if the test fails. Turn on this functionality by defining a test table and the property RefreshMinutes or TestFrequencySeconds in the JDBCConnectionPool tag from your config.xml file.

Refresh runs asynchronous to any client application code, and will temporarily reserve all currently unused pool connections to test. It will reserve all those connections during the complete test time. If new application requests for a connection come in during this time, and if InitialCapacity is smaller than MaxCapacity and fewer than MaxCapacity connections are open currently, every connection request that comes in will open CapacityIncrement new connections until the maximum number of allowed connections in the pool is reached. (This situation can lead to a connection peak effect as there may be more connections opened than are being used simultaneously.) If a maximum number of connections are open already, ResourceExceptions will be thrown.

Consider carefully whether the refresh functionality is needed for your JDBC pool. One very useful case for refresh functionality is if there is a firewall between WebLogic Server and the database, which closes idle socket connections. We'll discuss this shortly.

If needed, alternative and better options for testing and refreshing connections are to set the TestConnectionsOnReserve property to true, which ensures that every connection that is requested from the pool will be tested before it is forwarded to the application code. If the test fails, it will be reopened automatically. If the database was temporarily unavailable or down, the connection pool can be completely refreshed by using the weblogic.Admin RESET_POOL command, which ensures that all connections are refreshed, whereas the refresh functionality only refreshes unused connections.

Extended Startups

Bad performance of the database or network can be the reason why connection requests to the underlying database take a very long time, leading to very long startup times for the WebLogic Server. During WebLogic Server startup, the InitialCapacity property in the JDBCConnectionPool defines the amount of connections that are created during JDBC connection pool initialization. If the creation and initialization of the underlying physical connections to the database take a long time, the start of the WebLogic Server instance will also take a very long time.

If connection requests to your database generally take a long time and you would not like WebLogic Server to take this long during startup, set InitialCapacity to 0. This setting will create a pool with no physical connection during startup. If you would like to have all connections created during the first connection request, set CapacityIncrement to the full amount of connections available in the pool. The first request will take a very long time, but after that the connection pool will be fully operable.

Some JDBC drivers, like the BEA type-4 Oracle driver delivered with WebLogic Server 8.1 SP2 and later, have properties that limit the maximum amount of time a connection request waits. For these drivers, specify the property LoginTimeout in the Properties attribute for JDBCConnectionPool in the config.xml file. This value will be passed by WebLogic Server to the JDBC driver (see Resources online at www.javapro.com for a BEA support solution on this topic).

Incorrectly configured JDBC drivers will lead to errors or exceptions during JDBC pool connection creation. As mentioned previously, WebLogic Server will try to create InitialCapacity physical connections during WebLogic Server's startup. If the JDBC driver for the pool is not configured correctly or the database is not available during WebLogic Server startup, the initialization of the physical connections will not succeed.

In versions prior to WebLogic Server 8.1, a scenario resulted where the JDBC connection pool was not created successfully and could not be used afterward. Also, a refresh, drop, or recreate of the pool was not possible. The workaround here is to configure a JDBC pool that does not open any connections during startup (InitialCapacity="0"). A connection pool with no opened connections will be created, and every connection request to the pool will open CapacityIncrement connections after the database is available again or throw an error message if the database is still down.

Oracle RAC/TAF Configuration

There are some details you should consider for configuring JDBC connection pools for Oracle RAC/TAF. This information is provided in the "Oracle Real Application Clusters (RAC) Configuration and Testing" BEA support pattern. It contains information about how to install and configure Oracle RAC, some background information on common problems with Oracle RAC/TAF, and basic debugging tips and links to external resources.

Problems with databases or other resources that participate in distributed transactions can lead to an ORA-01591 error message in the Oracle database. An upcoming pattern "Investigating ORA-01591 Error Messages" will provide information on analyzing this issue (see Resources online at www.javapro.com).

In WebLogic Server 8.1, this behavior has changed. If there is a problem during JDBC pool creation, a JDBC pool with 0 initial connections is created. The configuration can be changed while WebLogic Server is still running. After the pool is configured correctly or the database is back up, the connections in the pool can be created and the pool can be used in the application. (See Resources online at www.javapro.com for information on how to configure JDBC connection pools for different drivers, including third-party drivers.)

Specific Solutions

Here are some typical error messages from incorrectly configured JDBC pool drivers. (See the sidebar, "Oracle RAC/TAF Configuration" for information on configuring connection pools for Oracle RAC/TAF.)

If you see an error message similar to this one, correct the specified user/password:

```
<05.11.2003 11.38 Uhr CET> <Error>
  <JDBC> <BEA-001150> <Connection
  Pool "myPool" deployment failed
  with the following error: 0:
  Could not connect to
  'oracle.jdbc.driver.
  OracleDriver'.

The returned message is: ORA-01017:
  invalid username/password; logon
  denied
```

It is likely that the login or password is not valid. It is also possible that something else is invalid in the configuration or that the database is not available.

If you see an error message similar to this one, correct the database name:

```
<06.11.2003 14.21 Uhr CET>
<Warning>
  <JDBC> <BEA-001129>
<Received exception while creating
  connection for pool "myPool":
E/A-Exception: Connection refused
  (DESCRIPTION=(TMP=)(VSNNUM=
  153092352)(ERR=12505)
(ERROR_STACK=(ERROR=(CODE=12505)
  (EMFI=4))))>
<06.11.2003 14.21 Uhr CET> <Error>
  <JDBC> <BEA-001150>
<Connection Pool „myPool"
  deployment
  failed with the following error:
  0:Could not create pool
  connection.
  The DBMS driver exception was:
E/A-Exception: Connection refused(
  DESCRIPTION=(TMP=)(VSNNUM=
  153092352)
(ERR=12505)(ERROR_STACK=(ERROR=
  (CODE=12505)(EMFI=4)))).>
```

If you see an error message similar to this one, check the tns entry or your environment settings like PATH or LD_LIBRARY_PATH:

```
####<Apr 17, 2003 1:58:44 PM CEST>
  <Error> <JDBC> <mydomain>
  <myserver> <main> <kernel
identity>
  <> <001060>
<Cannot startup connection pool
"myPool" weblogic.common.
  ResourceException:
  weblogic.common.
  ResourceException:
  Could not create pool connection.
  The DBMS driver exception was:
  java.sql.SQLException: Io
  exception: The Network Adapter
  could not establish the
  connection
```

On the Microsoft Windows platform, PATH needs to point to the client and OCI libraries; in Unix environments, LD_LIBRARY_PATH needs to point to the directories where the client installation copied the client and oci libraries.

If the tnsnames.ora file or ORACLE_HOME is not set correctly, this Oracle error will be thrown:

```
ORA-12154: TNS could not resolve
  service name
```

Ensure that your ORACLE_HOME environment variable points to the correct directory and the tnsnames.ora file is stored in the correct directory. Also verify through sql-plus if a connect to this database is successful. (See Resources online at www.javapro.com for additional information on ORA-12154 and related Oracle SQL errors.)

A related error message is ORA-24327, which is caused most likely by a configuration problem:

```
LOGIN ERROR CODE: 24327
  <Error> <JDBC Connection Pool>
  Cannot startup connection pool
  "xxxPool" weblogic.common.
  ResourceException:
  Could not create pool connection.
  The DBMS driver exception was:
  java.sql.SQLException: ORA-24327:
  need explicit attach before
  authenticating a user.
```

Incorrect locale settings may lead to error messages like this one:

```
weblogic.management.
  DeploymentException: Error creating
  connection pool myConnectionPool:
  0:Unable to load locale
  categories
```

Ensure that you have set the correct locale environment before starting WebLogic Server. Double-check by starting a database client in the same environment, and check if your locale works there.

If the database was intermittently down, a connection reset or refresh will happen if the TestConnectionsOnReserve property is set to true and the connection test query fails. You will find related messages in the WebLogic Server log file, similar to this one:

```
ORA-03113 end-of-file on
  communication channel and/or
  ORA-01012 not logged on:
```

This message is for your information only, and, as long as the connection could be created successfully during the refresh or reconnect, it is harmless (see Listing 1 online at www.javapro.com).

Database Dilemmas

If the configured amount of allowed open cursors in an Oracle database is exceeded, this error message will be thrown:

```
java.sql.SQLException: ORA-01000:
  maximum open cursors exceeded
```

See What's New at the 2006 JavaOne℠ Conference

Come to the 11th Annual JavaOne℠ conference and see where the Java™ platform is headed. More than 300 technical sessions and BOFs equip you with the knowledge you need for your current and future technological innovations.

Hear from industry experts, including Graham Hamilton and Bill Shannon of Sun Microsystems, as they discuss key directions for the next releases of the Java platform.

Plus, enhance your development skills in hundreds of expert-led sessions in five tracks over four days:*

: Track One :
Java Platform, Standard Edition (Java SE)

: Track Two :
Java Platform, Enterprise Edition (Java EE)

: Track Three :
Java Platform, Micro Edition (Java ME)

: Track Four :
Tools

: Track Five :
Cool Stuff

the POWER of JAVA™

GET CONNECTED
with the JavaOne Conference Event Connect tool.

As a Conference attendee, you can now connect with experts and fellow developers before the event. This effective online solution allows you to quickly schedule meetings with industry leaders and interact with other attendees.

Register for the Conference and get connected today.

REFER YOUR FRIENDS
and receive an iPod nano music player!

One iPod nano music player per qualifying registrant. While supplies last.

SAVE $100!**
REGISTER TODAY
at java.sun.com/javaone/sf

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

PLATINUM COSPONSORS

bea Think liquid. IBM ORACLE

GOLD COSPONSORS

JUSTSYSTEMS NOKIA Connecting People SAP TERRACOTTA

SILVER COSPONSORS

Adobe QUEST SOFTWARE Sprint Together with NEXTEL

Sun microsystems

JavaOne℠ Conference | May 16–19, 2006
JavaOne℠ Pavilion: May 16–18, 2006, Moscone Center, San Francisco, CA

JavaOne
Sun's 2006 Worldwide Java Developer Conference

There can be several causes for this error. In one case, the prepared statement cache is configured to be too large. Check the configuration of your JDBC pool regarding the prepared statement cache. Every prepared statement will use one open cursor in the Oracle database. The statement cache holds prepared statements on a connection basis, which means that the Oracle database will use up to:

```
(StatementCacheSize) x
(MaxCapacity)
```

open cursors for every configured pool. Since open cursors will be used for other objects also (for example, stored procedures or result sets) the number of open cursors needs to be configured high enough to hold all the statements in the statement cache. The setting for OPEN_CURSORS is per session/connection. (See Resources online at www.javapro.com for additional information on statement cache configuration.)

Some versions of Oracle drivers (thin or oci) have a cursor leak in the XA driver class (oracle.jdbc.xa.client.OracleXADataSource) that leads to the ORA-01000 error message after some time. Ensure that on the database side the DBA_PENDING_TRANSACTION view has the correct permissions:

```
grant all on
  DBA_PENDING_TRANSACTIONS to
  public
grant all on
  DBA_PENDING_NEIGHBORS to public
grant all on
  DBA_2PC_PENDING to public
```

The cursor leak is fixed in Oracle version 9.2.0.5 and 10g. This known issue is described in Oracle Metalink Case 3151681 (see Resources online at www.javapro.com).

If you have configured a firewall between the database and WebLogic Server, and this firewall closes idle connections after a certain amount of time, the JDBC pool refresh functionality can be used to ensure that connections from the pool are not closed by the firewall. Here is a common error message thrown after such a closed connection:

```
java.sql.SQLException: ORA-03113:
  end-of-file on communication
  channel
  at weblogic.db.oci.OciCursor.
    getCDAException(
    OciCursor.java:240)
  at weblogic.jdbc.oci.Statement.
    executeQuery(Statement.
    java:916)
  at ...
```

This error occurs because the socket connection is considered okay from both the WebLogic Server and the database side; both may try to write into this socket connection and fail because it has been closed by the firewall without notification or error message to the participating parties. Use the refresh functionality to ensure that the connections are not idle long enough for the firewall to close them.

You can configure refresh functionality by setting the RefreshMinutes or TestFrequencySeconds property so that connections are tested at least one time during the idle period configured for the firewall. To enable the refresh functionality, the TestTableName property also has to be set (see Resources online at www.javapro.com for more information).

However, every Java Message Service (JMS) server takes one connection from the JDBC pool if a JDBC store is defined. This connection is considered as reserved by the pool so that the refresh functionality will not test and refresh those connections. Here is a typical error message:

```
JMSServer "myJMSServer", store
  failure while writing message for
  queue myQueue, java.
  io.IOException
```

This kind of situation can be solved either by sending at least one dummy JMS message during the idle period so that the firewall will not close the connection, disabling the connection closure by the firewall, or defining a separate JDBC pool that will be used as a JDBC store for JMS servers and use weblogic.Admin RESET_POOL to reopen the connections at least one time during the idle period.

Stay Fresh

This problem has been addressed in later WebLogic Server versions (6.1 SP7, 7.0 SP3, and 8.1 SP1 or later). If JMS is idle, the database is pinged every five minutes to keep the connection fresh and prevent the firewall from closing these connections.

If getVendorConnection() is used to receive the underlying physical connection for a JDBC pool connection, the RemoveInfectedConnectionsEnabled property's setting needs to be checked.

Some advanced JDBC commands require the physical connection as an argument. To be able to do this, getVendorConnection() can be called to get the physical connection used by the connection pool. However, the application code that uses the physical connection needs to be implemented carefully to avoid follow-up problems. As the JDBC connection pool is implemented to ensure as much security and availability as possible, any connection for which getVendorConnection() was called is refreshed automatically after the application's usage (RemoveInfectedConnectionsEnabled="true").

Thus, connection pooling loses its effect, that is, every connection is closed and reopened after usage. Consider carefully if the application code changes or destroys something on the physical connection that makes a reopen necessary. If and only if this effect is not the case, set the RemoveInfectedConnectionsEnabled property to false (see Resources online at www.javapro.com for more information).

As type-2 JDBC drivers use native code, problems in these drivers may lead to a crash of the JVM and WebLogic Server. If your server crashes, check information provided in the BEA support pattern, "Binary Core File Analysis Pattern" (see Resources online at www.javapro.com). This information will help you to track down the native library that causes the crash and provides tips on how to solve the problem.

A JDBC connection uses one execute thread from WebLogic Server to perform its work, which means that a hanging request to a database blocks one thread in WebLogic Server. Problems with the JDBC connection or the database infrastructure can lead to a hang in WebLogic Server or the application.

Information on analyzing this kind of problem is available in a BEA support pattern, "JDBC Causes Server Hang" (see Resources online at www.javapro.com). General information on how to track down a hang situation in WebLogic Server is already available in the BEA support pattern, "Generic Server Hang Pattern.

A memory leak of JDBC objects leads to an OutOfMemoryError or growing process size. JDBC objects used by connections from the JDBC connection pool or in application code—that connect directly to a database—are part of the heap or native memory of the process. If these objects are not closed and freed correctly, a memory leak is the consequence. It will cause increased heap usage or growing process size, and it will lead to an OutOfMemoryError after the JVM or the operating system cannot provide any free memory anymore.

If you see OutOfMemoryErrors in your system and suspect JDBC objects to be the cause, please check the "Investigating Out Of Memory/Memory Leak Problems" pattern, which provides general information on how to analyze memory leak problems. If you detected connections-leak error messages in your WebLogic Server log file or through JDBC pool monitoring in your administration console, the BEA support pattern "Investigating JDBC Connection Leaks" provides troubleshooting for this issue. This pattern is being developed at the time of this writing. A link will be provided on the BEA support patterns page as soon as it is available.

Flagging Issues

JDBC debugging and tracing sometimes is key for finding out what is going on and analyzing the SQL statements that are sent to the database. However, JDBC is a multilayered subsystem, of which only parts are inside WebLogic Server. Debugging and tracing of the JDBC driver layer is highly driver-dependent. Information regarding debug and trace flags for the drivers is available from the driver vendors. At the WebLogic Server side, different JDBC debug flags are available.

jDriver JDBC tracing can be switched on through the WebLogic Server administration console, or in the config.xml file directly by setting <JDBCLoggingEnabled> in the Server tag (see Resources online at www.javapro.com for more information). The ServerDebugMBean has some JDBC-related flags that can be turned on in the config.xml file. Put a new <ServerDebug> tag into the <Server> tag from the WebLogic Server instance you would like to debug:

```
<Server Name="myserver" >
  <ServerDebug Name="myserver"
    JDBCConn="true" JDBCSQL="true"
    JTAJDBC="true" />
</Server>
```

Alternatively, these debug flags can also be set as system properties during WebLogic Server startup:

```
-Dweblogic.Debug=weblogic.
  JDBCConn,weblogic.JDBCSQL,
  weblogic.JTAJDBC
```

These debug flags and tracing can be very verbose, so consider very carefully where you turn on those flags. They will create a lot of output and also possibly have a performance impact on your system. They should not be turned on in production systems.

Guideline	Description
Set InitialCapacity = MaxCapacity	Ensures that all connections are opened during WebLogic Server startup. Since creation of a physical database connection is very expensive, all needed connections should be opened immediately and kept open.
Disable shrinking by setting ShrinkingEnabled to false.	As mentioned previously, creation of physical database connections is expensive; therefore, connections should be established once and kept during the complete lifetime of the WebLogic Server instance.
Turn off refresh functionality if it is not needed.	Details for JDBC connection pool issues and firewalls were discussed previously.
Set TestConnectionsOnReserve to true.	Ensures that connections are tested before they go to the application and are reopened if needed.
Set TestConnectionsOnRelease to false.	Since connection testing is an overhead that should be avoided where possible, connection testing for connections that an application gives back to the pool is an unnecessary effort. As long as connections are tested during a getConnection(), this attribute is not needed.
Set the number of connections in the JDBC pool equally to the number of execute threads that use the connections.	This setting helps to avoid ResourceExceptions.

Table 1 | JDBC Connection Pools in Production Systems Administrators can use these general recommendations for configuring JDBC connection pools.

For drivers that do not allow turning on JDBC debugging or do not print enough information, installing a spy driver can help to debug all SQL statements between the JDBC driver and database. This driver implements an additional layer that prints debug information into a log file. You can download the P6Spy driver at its Web site, where you will also find related documentation (see Resources online at www.javapro.com).

WebLogic Server multipools can be used for high availability or load balancing purposes. Depending on the algorithm, the multipool behavior is different. The "Investigating JDBC Multipool Issues" pattern assists with this topic (see Resources online at www.javapro.com).

You can tune JDBC connection pools for production environments. Configuration of JDBC connection pools for production systems is a critical and important task to ensure stability and performance. The general recommendations in Table 1 may help as a starting point for administrators. JP

Maria Salzberger is employed in customer support with BEA where she supports WebLogic Server and WebLogic Integration, primarily JDBC and transactions, EJBs, core server performance, and stability. Maria is also a contributor to the BEA Support Pattern initiative (**http://support.bea.com/application_content/product_portlets/support_patterns/wls/wls_support_patterns.jsp**).

One or the Other

Best of breed or high level of integration? History is on the side of the universal platform

by Peter **VARHOL**

Among application life-cycle professionals, there has been a debate that has raged for over a decade on the appropriate approach toward the acquisition and use of software tools. Simply put, that debate is whether it is better to use tools that are "best of breed," or those that offer a high level of integration with each other. The implication is that a vendor with integrated application life-cycle tools doesn't necessarily offer the best point solutions for specific tasks, while those who do offer the best individual tools lack a broad range of solutions.

There is some truth to this assumption, based on both logic and practice. Vendors (in the case of commercial products) and small teams of developers (for open source or freeware projects) will tend to have a greater specialized expertise in an area that results in the development of a single tool with deep analysis and narrow breadth.

Less obvious is how a commercial vendor, especially a larger and successful one, can fail to produce a set of tools that is both well integrated and in-depth. However, the inference is still generally correct. That larger vendor has to make more money to support a larger development staff (along with all of the sales and support functions that go along with a large company), and simply lacks the resources to offer in-depth analysis at every point in the application development life cycle.

There's more to this than meets the eye, too. It really doesn't pay, in an economic sense, for any one vendor to be the best of everything. The key is to be just good enough to maximize sales without expending excess resources in doing so, which seems to lead to the conclusion that the individual best-of-breed point solution is the better alternative, but that conclusion is misleading too. The question is whether the integration into a single user framework, and the sharing of data between tools, is of more value than the collective power of the individual best-of-breed tools.

No Easy Choice

The debate between best-of-breed and integrated solutions continues. However, there's a larger issue: how does one identify which solutions are which? Certainly marketing hype doesn't help. Most life-cycle tools are happy to call themselves best of breed, since there are no criteria for defining just what that means. You may spend as much time and effort figuring out which tool delivers on the best-of-breed promise as you do using the tool in the application life cycle.

To a large extent, best of breed is a matter of personal preference. The needs of most development teams are at least partially unique, and no single tool is best across the range of those needs. For example, a thread analyzer might be able to detect thread deadlock, but only tracks two given threads at any one time. A second, similar thread analyzer can track any arbitrary number of threads, but can't detect deadlock among any of them. Both are useful, but their usefulness is different in different development or deployment situations.

Nevertheless, there are some characteristics in a tool that most users will be able to agree are more powerful than others, and it may be possible to get some level of agreement as to what those features are, and what tool is best at them. But it's almost never that clear cut. Some tools may have slightly different purposes, with differing feature sets, and others may have fewer features, but better usability. It is possible to pick one in each category that best serves the needs of a specific set of users and applications, but not one single best one.

In addition to this measure of subjectivity, individual application life-cycle tools have some significant disadvantages. They can't share data with one another, making it more difficult to use together to solve specific problems. They also require separate knowledge on how to use each tool because there is unlikely to be any common keystrokes or actions that can be leveraged across them.

Go Online

Visit www.javapro.com for related resources. Simply type the Locator+ code into the field in the upper-right corner of the page.

Download

JP0511 Download all the code for this issue.

Discuss

JPTALK Discuss this article in the "Talk to the Editors of *Java Pro* Magazine" discussion forum.

Read More

JP0511OE_T Read this article online.

JP0407OE_T Read the related article "Think Integration" by Peter Varhol.

JP0406PV_T Read the related article "Building a Better Application Life Cycle" by Peter Varhol.

FOSRLM_T Read the related article "Design for Success" by Edmund X. DeJesus.

However, there is a level above the individual point tool. If a tool has the same look and feel as other tools across the application development life cycle, and if those tools are able to share some data with each other and make use of that data in analysis, then these tools together consist of a suite. A *suite* makes it at least theoretically possible to leverage the power of the individual best-of-breed tools, while also leveraging the data they create.

Unfortunately, there are no standards today for sharing data and using shared data. The vendors attempting this strategy either simply bundle tools together with little or no data sharing, or they use proprietary methods of sharing some data. In some ways, the latter is worse than the former because the vendor has in effect made its own decisions on what data is important, and when it needs to be accessed. This assessment might be true if all application life-cycle professionals worked in the same way on the same applications, but that's not at all the case.

How Suite It Is

The traditional development manager's role is likely to remain similar in the modeling team because plans and communication remain a high priority. Schedules, resource allocation, direction, and status requirements probably won't change significantly, no matter what project technologies are employed.

Still, suites are a step forward from point tools because they are more accessible than they are as individual tools. Because the tools are more accessible, they make the user more productive. Rather than launching individual tools and switching back and forth, having to cope with the differences in user interfaces and looking at the resulting data in different measures and formats, suites provide a measure of common ground. It is not ideal, but it is an improvement.

Beyond the suite, there is the integrated platform, which hosts multiple tools in a common framework. Some of the more modern IDEs are good examples of the state of the art in integration. The tools still exist, but might now be called features. Unlike tools, however, they all have the same look and feel, defined by the underlying platform. They show results in similar terms and formats, and can exchange data to provide more complete analyses.

And history is on the side of this integrated solution, assuming that the usability is consistent and data can be shared between features. I have been a user of separate compilers, editors, and debuggers early in my development career, and at that time there were definite opinions as to the best of each. Yet outside of open source, these basic tools are almost unknown today. Development tools have largely merged into suites or full-fledged integrated platforms.

That merging represents forward progress, in both productivity and usability, and that is why the integrated platform is the inevitable. Developers and other application life-cycle professionals swore by their tools of choice (I merely swore at them), but any productivity at all required extensive study and use of the tool itself, which detracted from the real work with the application.

It is too early to say that the specific platform will be Eclipse; although, it has made a good start in this direction. Thanks to a robust plug-in architecture, and an active solicitation of plug-in tools across the entire application life cycle, Eclipse might be the ultimate application life-cycle platform. The look and feel of many of these plug-ins are similar because plug-in developers tend to use Eclipse features and follow the Eclipse plug-in guidelines.

For those whom individual tools remain an attractive option, is there an answer for the continuation of best-of-breed point products? Yes, but only if they adopt standards for both look and feel and data sharing. The former means that there should be some agreement on Java user interface toolkits and conventions surrounding menu layouts and accelerator key use.

The latter need, enabling data sharing between different point products whether from the same vendor or from different (and often competing) vendors, can be addressed by implementing these products as separate Web services, rather than stand-alone applications. Data can be shared through a publish-subscribe strategy; if a specific data item is available from Tool A, then Tool B can make use of it. If Tool A isn't a part of the toolkit, then Tool B can work well without it.

Standard Value

Web services implementation has some interesting implications in tool architecture and design. It is likely that tool developers will write separate user interfaces and analysis engines, so that users can swap out the provided user interface for one of their own. This feature in effect customizes the analysis for their specific need, providing at least some of the value of a more integrated platform. In fact, for those who can write their own user interfaces, it goes beyond the integrated platform for that specific area, providing users with just the information they need, and in the format they need it.

If the application life-cycle tools industry can be successful in establishing broad use of tools among application life-cycle professionals, it must devise base standards that all adhere to. These standards include Web services interfaces to tools and published result formats to be able to create integrated sets of individual tools with user interfaces focused on bringing together the information from customized sets of tools. In time, this standardization might end up being more valuable than the integrated platform. JP

> Most life-cycle tools are happy to call themselves best of breed, since there are no criteria for defining just what that means

Peter Varhol is a senior member of the technical staff for Progress Software. Contact Peter at **peterv@mv.mv.com**.

Put a Plug-In to Use

by Kevin **JONES**

You have a plug-in and its JFace-built UI. It's time to combine techniques to create a JDBC plug-in

In recent columns I've discussed how to start writing a plug-in using Eclipse's plug-in wizard and how to use JFace, which is a framework for building UIs that is part of Eclipse. Now let's look at how to back fit the code demonstrated in my column "Take a JFace Detour" to the plug-in we discussed creating in my column "Get Acquainted with Eclipse Plug-Ins" (see Resources online at www.javapro.com for links to both articles). Along the way we'll also take a brief look at layout managers, in particular GridLayout; event handlers; and logging within a plug-in.

The initial plug-in consisted of four classes. Two were the class that represented the plug-in—JdbcviewerPlugin—and a class that represented the view—JDBCTableView. The view class contained two nested helper classes, ViewContentProvider and ViewLabelProvider, and all in all the code was not particularly complicated. Now let's extend this plug-in to allow the user to specify the JDBC driver to load, JDBC URL to use, and SQL statement to select, as well as execute this JDBC command. We need to rewrite the JDBCTableView class to accomplish these goals.

> Luckily, Eclipse has a mechanism to display errors: the Error log

Recall that it is the createPartControl() method that is called by Eclipse to allow the plug-in to create its UI. In the current plug-in the display is very simple (see Figure 1). The new plug-in will have a more complicated display and layout (although not too complicated).

The initial UI allows us to enter the information needed by the JDBC code, notably the name of the class of the JDBC driver, the JDBC URL, and the select statement. Clicking the Go! button executes the query, with the table showing the resulting data. This UI is laid out in grids by using multiple GridLayouts and composites.

There are several things to note in the code to create the JDBC entry fields (see Listing 1). We are using Composite, which is a UI widget that can contain other widgets, and it is similar to the Panel in AWT or JPanel in Swing. We use nested composites with each composite containing the necessary widgets, which allows for a great deal of flexibility in the layout.

Widget Layout

Like AWT and Swing, the JFace/SWT combination also uses layout managers to lay out the widgets correctly. In this case we are using the GridLayout exclusively. The GridLayout, like its name suggests, allows us to lay out widgets in a grid. To use this

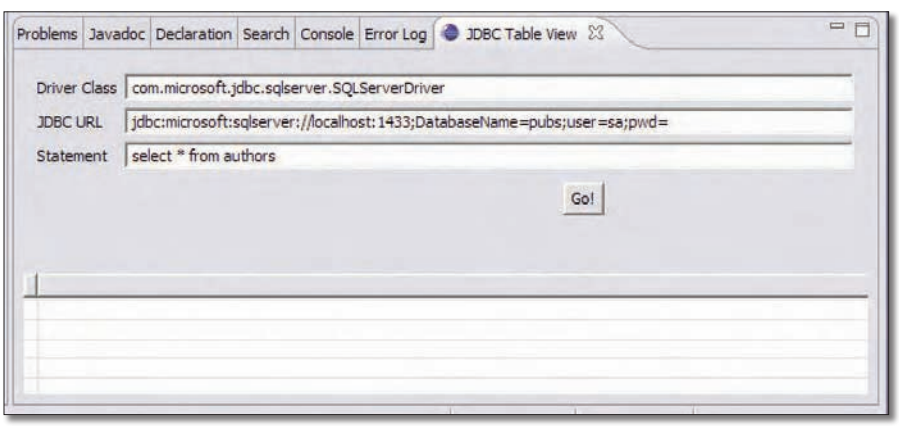

Figure 1 | Simple UI The createPartControl() method called by Eclipse allows the plug-in to create its UI, which in its initial form is very simple.

Go Online

Visit www.javapro.com for related resources. Simply type the Locator+ code into the field in the upper-right corner of the page.

Download

JP0511 Download all the code for this issue.

JP0511PI Download the code for this article separately. The code demonstrates building a JDBC Viewer plug-in for the Eclipse platform.

Disuss

JPTALK Discuss this article in the "Talk to the Editors of *Java Pro* Magazine" discussion forum.

Read More

JP0511PI_T Read this article online.

JP0507PI_T Read the related article "Take a JFace Detour" by Kevin Jones.

JP0506PI_T Read the related article "Get Acquainted with Eclipse Plug-Ins" by Kevin Jones.

JP0505PI_T Read the related article "Eclipse SWT 101" by Kevin Jones.

Listing 1 Create Entry Fields

```
Composite top = createTopComposite(_mainPanel);

createJDBCComposite(top);

private Composite createTopComposite(
  Composite composite)
{
  GridData gd = new GridData(
    SWT.FILL, SWT.FILL, true, true);
  Composite top = new Composite(composite, SWT.NONE);

  top.setLayout(new GridLayout(1, false));
  top.setLayoutData(gd);
  return top;
}

private void createJDBCComposite(Composite parent)
{
  Composite widgets = new Composite(parent, SWT.NONE);
  GridData gd;
  gd = new GridData(GridData.FILL_HORIZONTAL);

  widgets.setLayout(new GridLayout(2, false));
  widgets.setLayoutData(gd);

  Label label = new Label(widgets, SWT.NONE);
  label.setText("Driver Class");

  gd = new GridData(GridData.FILL_HORIZONTAL);
  _driverName = new Text(
    widgets, SWT.SINGLE | SWT.BORDER | SWT.H_SCROLL);
  _driverName.setText(_driverClass);
  _driverName.setLayoutData(gd);

  label = new Label(widgets, SWT.NONE);
  label.setText("JDBC URL");

  gd = new GridData(GridData.FILL_HORIZONTAL);
  _jdbcURL = new Text(
    widgets, SWT.SINGLE | SWT.BORDER);
  _jdbcURL.setText(_connectionString);
  _jdbcURL.setLayoutData(gd);

  label = new Label(widgets, SWT.NONE);
  label.setText("Statement");

  gd = new GridData(GridData.FILL_HORIZONTAL);
  _statement = new Text(
    widgets, SWT.SINGLE | SWT.BORDER);
  _statement.setText(_query);
  _statement.setLayoutData(gd);
}
```

In the code to create the JDBC entry fields we use Composite, a UI widget that can contain other widgets and that is similar to the Panel in AWT or JPanel in Swing. Nested composites contain the necessary widgets that provide a great deal of flexibility in the layout.

Listing 2 Button, Button

```
private void createButtonComposite(Composite top)
{
  Button go = new Button(top, SWT.PUSH);
  GridData gd = new GridData(
    GridData.HORIZONTAL_ALIGN_CENTER);
  go.setLayoutData(gd);
  go.setText("Go!");

  go.addSelectionListener(new SelectionListener()
  {
    public void widgetSelected(SelectionEvent arg0)
    {
      _table.dispose();
      addTableViewerControl(_mainPanel);
      createJDBCTableModel();
      _mainPanel.pack();
      _mainPanel.getParent().pack();
    }

    public void widgetDefaultSelected(
      SelectionEvent arg0)
    {}

  });
}
```

Because we have to know when the Go! button is clicked, which requires an event handler, the code for the Go! button is different than before. After creating the button we add a selection listener to it. The SelectionListener event is fired when the button is pushed.

layout manager we specify how many columns will be used for the layout (the second constructor parameter specifies if the columns should have the same width). As widgets are added to a GridLayout they are added left to right, top to bottom. To help with layouts, the GridLayout has an associated GridData class that is not attached to the GridLayout explicitly, but it is instead associated with a widget that is added to the grid. It is the GridData that gives the layout its power. Using the GridData you can specify such attributes as alignments, fills, and sizes.

The top composite simply creates a composite that uses a GridLayout. The GridLayout contains a single column. We also associate a GridData with the composite. This GridData specifies that the composite should fill as much horizontal and vertical space as is available to it.

The JDBCComposite is similar, but now the grid contains two columns. We put the labels in the leftmost column and the edit fields in the right-hand column. In this example the edit fields are initialized with hard-coded strings.

The code for the Go! button is different because we have to know when the button is clicked, and to do that we require an event handler (see Listing 2). This SWT/JFace code looks very similar to the equivalent SWT/Swing code. We create the button and then add a selection listener to it. There are various events we could handle for the button, but the one we want is the SelectionListener. This event is fired when the button is pushed. In the listener we get rid of the current table (and its associated view), recreate the table and the model, and then associate the new model with this view.

We now need to go through the export process again as was demonstrated in the first part of this series (see Resources online at www.javapro.com for a link to the column "Get Acquainted with Eclipse Plug-Ins," *Java Pro*, Vol. 9, No. 4). In our project (which is available for download at www.javapro.com) I have a Jars directory that contains the three JAR files that make up the Microsoft SQL

Listing 3 Error Tracking

```
private void createJDBCTableModel()
{
  JDBCTableModel model;
  try
  {
    if (_driverName.getText().length() != 0
    && _jdbcURL.getText().length() != 0
    && _statement.getText().length() != 0)
    {
    model = new JDBCTableModel(
      _driverName.getText(),
      _jdbcURL.getText(),
      _statement.getText());
      applyModel(model);
    }
  }
  catch (Exception e)
  {
    JdbcviewerPlugin.log.log(
      new Status(Status.ERROR,
      "com.develop.kevin.jdbcviewer",
      1,
      "Error Loading JDBCTableModel",
      e));
  }
}
```

A reference to the log from the JdbcviewerPlugin class calls its log() method, which takes a single Status object parameter. Log messages are sent to log listeners and in particular to the error log window.

JDBC driver. When you export the plug-in, you'll find these JARs included in the plug-in JAR file. These JARs are then included in the plugins classpath, which means that currently only JDBC drivers included in the plug-in's JAR file will work with the plug-in. Obviously this setup is too limiting, and we need to fix this problem.

Once the plug-in has been exported we need to get it referenced by Eclipse, which is where you may hit a problem. Eclipse caches plug-ins between restarts, and it stores data about the plug-ins in the \home\kevinj\dev\eclipse\configuration\org.eclipse.osgi directory. To get the new version of the plug-in loaded, you have to shut down Eclipse, copy the plug-in we created previously to the %ECLIPSE_HOME%/plugins directory, and delete the .bundle.##, .lazy.##, and .state.## files from the \home\kevinj\dev\eclipse\configuration\org.eclipse.osgi directory, where ## represents a series of digits. Then you should be able to restart Eclipse and see the latest version of the plug-in, which we can now view and execute the code (see Figure 2).

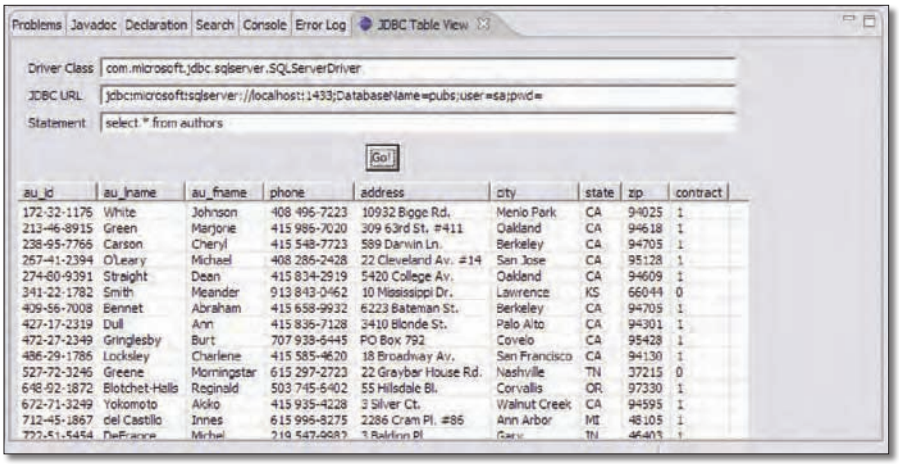

Figure 2 | Visible Execution When you restart Eclipse, you'll see the latest version of the plug-in.

Log Errors

Let's look at what happens when a plug-in hits an error. Luckily, Eclipse has a mechanism to display errors: the Error log. A reference to the log is provided through the PlugIn class. Recall that our JdbcViewerPlugin extends AbstractUIPlugin, which in turn extends Plugin. Plugin has a getLog() method. Help information for this method includes this description and recommendation: "Returns the log for this plug-in. If no such log exists, one is created."

This statement makes the log a singleton, and the best way to handle it is to create the log when the plug-in is first created and to store a reference to the log in the JdbcViewerPlugin class:

```
public static ILog log;

public JdbcviewerPlugin()
{
  super();
  …
  log = getLog();
}
```

We can now log errors whenever a problem occurs, such as when an exception is thrown. Looking at our JDBCTableView class we see one place where an exception is caught (see Listing 3). Here we get a reference to the log from the JdbcViewerPlugin class and call its log() method. The log() method takes a single parameter: a Status object. The Status constructor takes four parameters: the plugin ID, an error code, an error string, and a reference to a Throwable (if there is one). Log messages are sent to log listeners and in particular to the error log window.

In creating this JDBC plug-in we have briefly touched on SWT event handlers and layout managers to get a viewer where we can specify the JDBC information necessary to execute a query, and we now have a working viewer. Almost.

The primary problem with the viewer is that we are limited to the JDBC drivers we can use. Currently, we have to include the necessary JDBC JAR files with the plug-in. Since this arrangement is obviously not acceptable, we'll have to spend some time explaining how we get past this limitation. However, that's a subject for another column. Stay tuned. *JP*

Kevin Jones is a developer who researches and teaches Java programming and explores HTTP and XML. He lives in the U.K. and works for Developmentor, a training company based in the United States and Europe that specializes in technical training on Java and Microsoft platforms. Contact Kevin at **kevinj@develop.com**.

Put some Insight into your Software Architecture

Software Architecture insight

As a software architect, both business needs and technology demands affect your decisions. You have to make strategic architecture decisions based on what's achievable today—with an eye to future growth and change.

That's where FTP Online's *Software Architecture Insight* helps you. Twice a month, this must-have e-mail newsletter gives you both technical perspective and actionable advice for building applications and enterprise solutions. You'll learn about important topics like:

- real-world SOA
- proven middle-tier strategies
- best practices
- modeling business processes
- architecting for scalability
- migration strategies
- much more!

Free newsletter:
Sign up today!

www.ftponline.com/channels/arch/

FTP FAWCETTE TECHNICAL PUBLICATIONS

Build Interactive Workflows

by Anbarasu **KRISHNASWAMY**
and **VIJAY MANDAVA**

WebLogic Integration gives you the ability to create, manage, and implement a workflow for evaluating expense reports

Today's agile business requires significant human-machine interaction. Corporations have invested millions of dollars to automate their business processes to achieve significant ROI, but only some have succeeded in that goal. A lot of business units in large organizations still use the "evergreen" killer-application—e-mail—to conduct business operations. As good a medium as e-mail is, using it alone to track project status and lost productivity is impossible. Companies today need to measure how to decrease cost and time to complete a transaction, which can be achieved by automating their business processes.

Gone are the days when automating workflows was considered rocket science. There are plenty of software and tools to implement human workflow today, including WebLogic Integration, which provides a powerful worklist module to enable developers to implement human interactions. Let's look at how to develop end-to-end human workflow using WebLogic Integration.

Human workflow applications vary widely, but they all share some common characteristics. They implement long-running business processes, data or forms are created initially by either the system or the user, a sequence of approval processes follows, and the approver may be an individual or part of a larger group of people who gets a notification through e-mail and/or alert message. If the approver is a group, a specific individual may decide to work on the item and choose to claim it. The approver reviews the associated data, may approve or reject the form, and may add a comment or modify the associated data. The user may reassign the task to another approver. Users may need different views of forms that are waiting for their approval, are approved, and are rejected. Some users like to see an audit trail of the process.

These characteristics seem to imply that any document management system that tracks the document and has the ability to assign the documents might satisfy these needs. However, most enterprise business processes need data prefilled from various enterprise data sources, need that data updated, and need it to interact with other systems, and they must be able to perform all of these tasks in a transactional manner. The typical document management

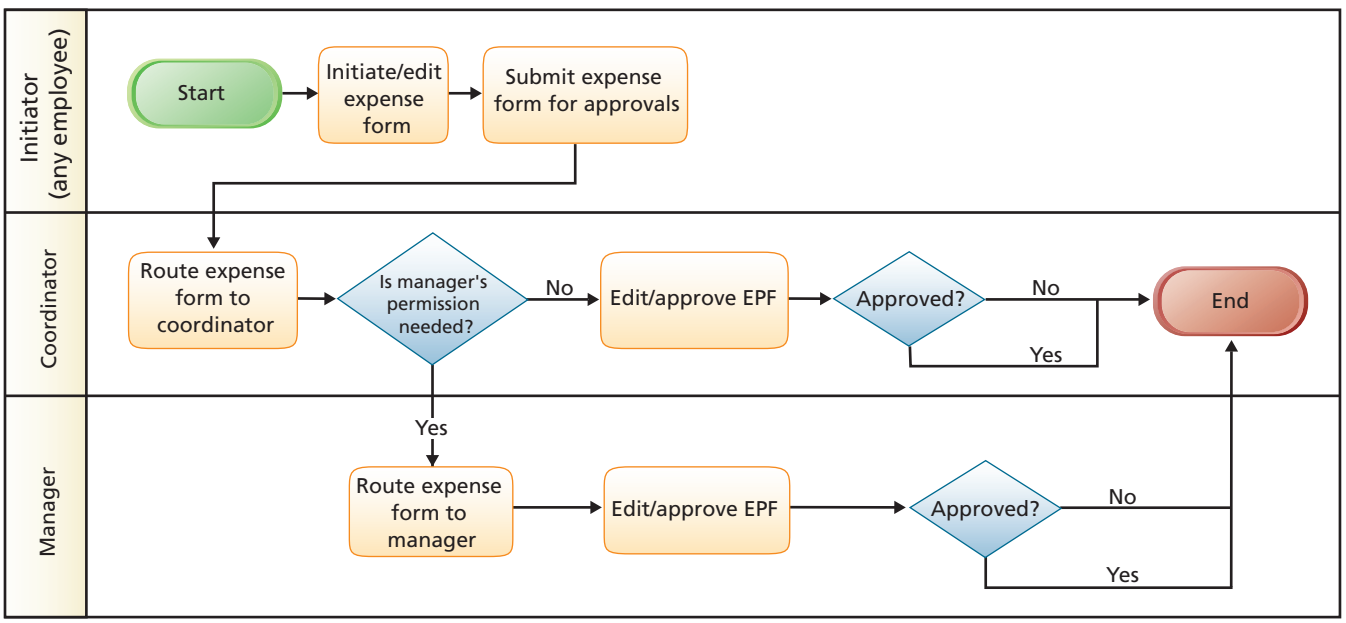

Figure 1 | Expense Report Process Flow This use case diagram shows an expense report's approval process. The business process represents an employee submitting an expense report that is routed either to a coordinator or a manager based on the amount of the expense. The employee can view the report's status at any point in the process.

systems satisfy only self-contained, all-encompassing document management.

Workflow Applications

WebLogic Integration provides BEA Process Definition for Java (JPD) processes and Worklist components to develop human workflow applications. The business processes are implemented using the JPD processes. The Worklist subsystem provides two primary controls: task control and task worker control. Task control represents a task itself, and tasks have these predetermined states:

- Assigned – a task has been assigned to a user or group but not yet claimed.
- Claimed – a user has claimed the task but has not started working.
- Started – a user has started working on it.
- Stopped – a user has stopped working.
- Completed – a task has been completed.

Tasks can be created by name or by using a task-creation XML document that provides more flexibility. Once created, the tasks can be assigned to users, groups, or both, and assigned tasks will go though their life cycle, changing their state. Tasks can also be assigned as overdue, which usually is used to perform escalation. The Worklist framework also provides callbacks to handle a task's complete, abort, or overdue events. Since these events are mutually exclusive, an event choice node is used to receive one of these events.

For example, let's look at an expense report application. Figure 1 shows the use case diagram of an expense report's approval process. The business process that we are building is that of an employee who will submit an expense report. The expense report is either routed to a coordinator or a manager based on the amount of the expense. The coordinator or manager can either approve or deny the expense report. At any point, the employee has visibility into the status of the submitted report.

The motivation for this application is to demonstrate how to build a custom work list and demonstrate the integration

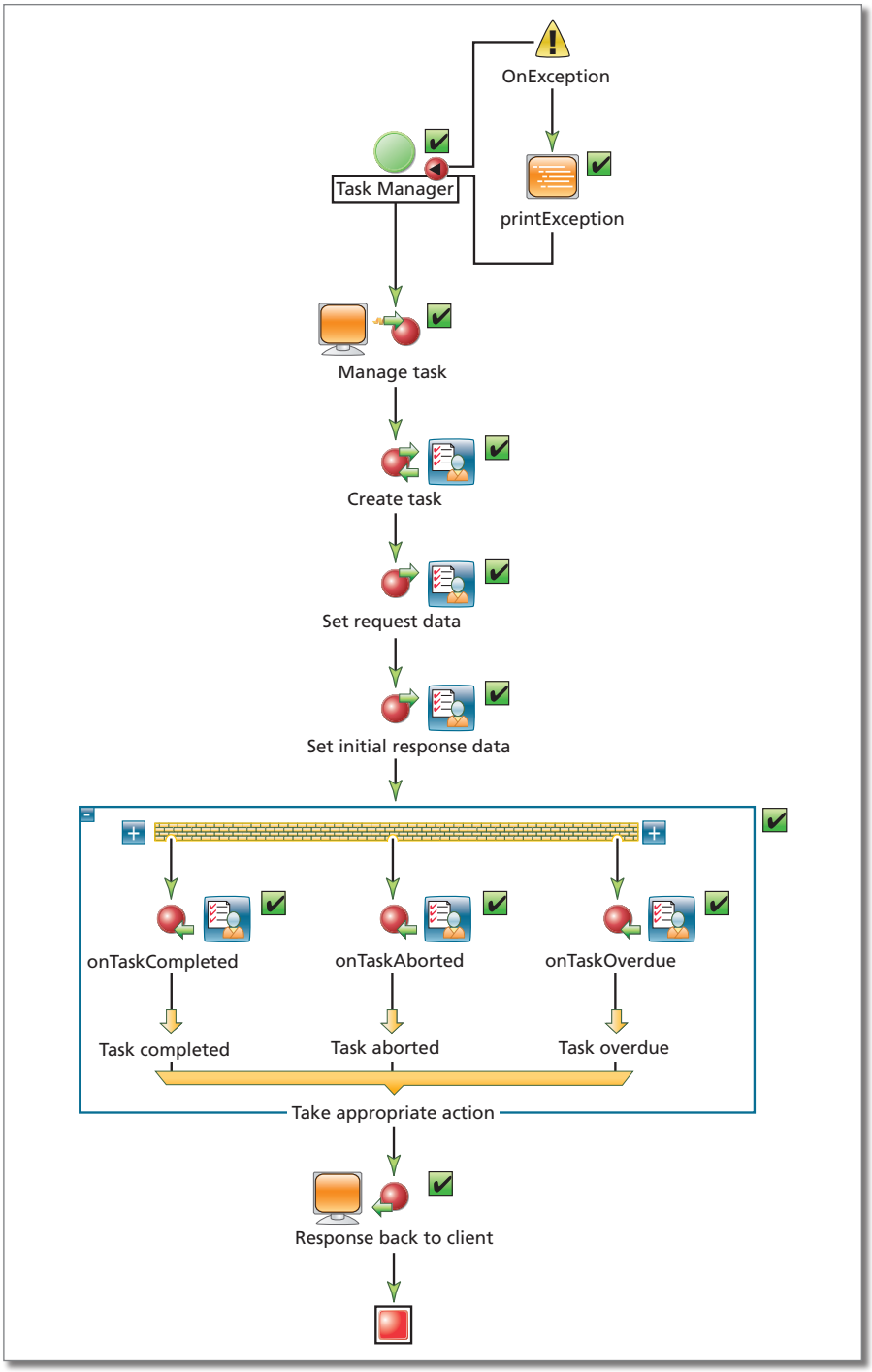

Figure 2 | Task Manager Process Simply right-click a JPD to convert a business process—for example, a TaskManager business process—into a control in Workshop.

Go Online

Visit www.javapro.com for related resources. Simply type the Locator+ code into the field in the upper-right corner of the page.

Download

JP0511 Download all the code for this issue.

JP0511KM Download the code for this article separately. This code includes the expense report worklist application.

Discuss

JPTALK Discuss this article in the "Talk to the Editors of *Java Pro* Magazine" discussion forum.

Read More

JP0511KM_T Read this article online.

JP041222DC_T Read the related article "Select the Best Persistence Architecture" by Doug Clarke.

JP0407KB_T Read the related article "Maintain a Healthy-Software Lifestyle" by Klaus-P. Berg.

EA0301AC_T Read the related article "Align Systems and Workflow With Process Integration" by Abby Christopher and Dan Ruby.

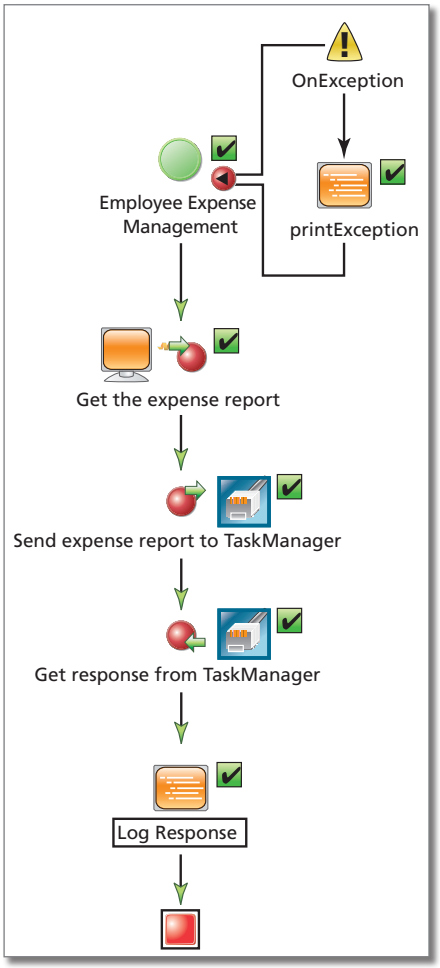

Figure 3 | Expense Management Process Complex business processes can add nodes that relate to their specific business functions. The entire population of the appropriate documents is done in the page flow.

among work lists and business processes. Let's discuss the steps to create the process. We'll start with schemas.

To build a loosely coupled system, documents must be passed so that the interface can be flexible. We created two XSDs for ExpenseSubmissionandforApprovalResponse. Workshop uses the XMLBeans technology, where if you drop an XSD file in a schema project, it will generate Java classes automatically for the XML documents.

The next step for creating our expense report application is to create the services. First, create a reusable task-manager process that can be accessed through a process control. The task control represents a task itself, while the task worker control is used to manipulate one or more tasks. Task control is used generally in JPDs to create and assign the tasks. The task worker control is used typically in user interfaces (page flow) and JPDs to access and update the tasks:

- Use the task control to create the task by name or by task creation XML.
- Assign the task to the user or group.
- Create an XML document that represents the form or user document.
- Set the request XML document (setRequestXML).
- Set the initial response document, if required.
- Assign any task properties. Task properties are any user-defined, string key-value pairs that can be used as a parameter for querying tasks.
- Create the three call-back handlers in an event choice node: OnCompleted, OnAborted, and OnOverdue.
- Return the response document.

Business processes should be developed so that they are modular. Reusable components for task management and e-mail management should be defined as independent business processes. The sample code includes a reusable task manager process that you can import into your project and use (see Resources online at www.javapro.com).

Process to Control

The generic TaskManager business process takes a TaskCreationXMLDocument request and sends the response back. The response can be reused by any business process. In our example, it is used by the empexpense business process. In this workflow, the data inside the TaskCreationXMLDocument is used to set the task name and set assignees. The expense report submitted by the employee is set as the request document, and a response document is sent with default values that the task assignee can update. With this approach, the shape of the response document is predefined.

For the TaskManager business process to be used by another business process, we have to convert it into a con-

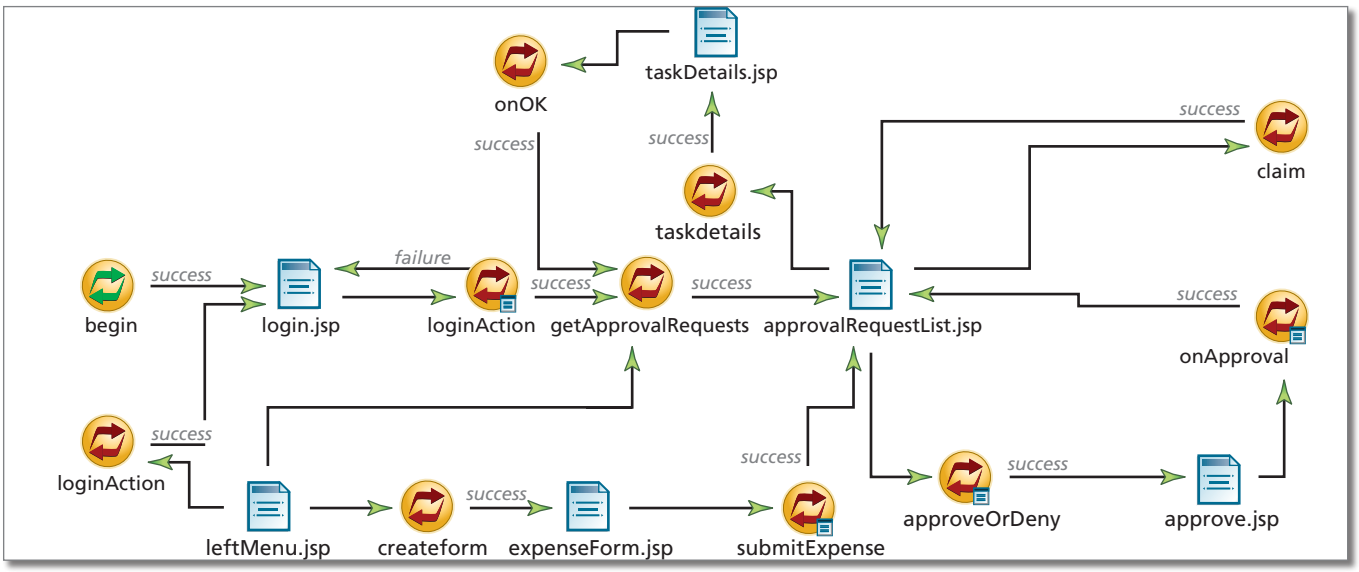

Figure 4 | Expense Management Page Flow Employees will see all of their associated expense reports that are associated with them when they log in. An association means all tasks that are claimed by, created by, and assigned to the users.

trol. Converting a business process to a control is a one-click operation in Workshop. It is done by right-clicking a JPD (see Figure 2).

Once you create the task manager process, you can create the process specific to your business requirements. In our example, we will create an employee expense report business process. This is a simple business process that just calls the TaskManager that has been exposed as a control. Complex business processes can add nodes that relate to their business. All the population of the appropriate documents is done in the page flow (see Figure 3).

We need to convert the empexpense business process into a control for these business processes to be called by a Web application. We created a controls project for the two controls mentioned previously. Building a controls project generates a JAR file and is placed in the Libraries folder, which will enable any other project within this application to use these controls.

The worklist sample application is an out-of-the-box Web front end to view and complete tasks. It can be accessed using the URL <your WLI server>:<port>/worklist, or it is available as a menu option under Workshop. The source code for the default worklist application is available online at www.javapro.com. You could either use this application and start customizing it to your needs or use this application as a reference on the usage of the API. In our case we used a hybrid approach of customizing some JSPs and using the worklist code as a reference on the usage of the API.

The next step is to create the user interface. Create a page flow for your application that represents the sequence of steps in your approval process. Create the JSP pages and actions nodes as necessary, and link them in the page flow. All these steps are similar to a regular Web application. The controller instantiates a task worker control and empexpense control. The task worker control has APIs to manage tasks, select tasks, and drill-down on details of a specific task.

The user-interface sequence is: the employee logs in, creates a new expense form, and submits the expense form. The manager/coordinator logs in, claims an expense report assigned to him or her, and approves/denies it. When employees log in, they will see all expense reports that are associated with them. *Association* means all tasks that are claimed by, created by, and assigned to the users. Figure 4 shows the page flow for the expense report application.

The Worklist Subsystem

The action methods in the page flow use the classes in the worklist subsystem such as TaskSelector and TaskWorkerControl. Here are some code snippets on getting all tasks claimed by a user, claiming a task, and completing the task.

You can define the criteria to query the tasks using the TaskSelector. It provides parameters to customize the query (see Resources online at www.javapro.com). To get all the tasks that a particular user has claimed, TaskSelector is used with the task worker control:

```
TaskSelector selector =
  new TaskSelector();
  // user as claimant
  selector.setClaimants(
    new String[]{username});
TaskInfo[] infos =
  worker.getTaskInfos(selector);
```

The task selector object also provides the query functionality required to query the tasks. Querying all tasks requires the user performing the oper-

Worklist Development Tips

Here are some best practices and tips for using WebLogic Integration Worklist that can help you develop the application right the first time.

- **Create a reusable task handler.** In a service-oriented architecture, you create loosely coupled, reusable, and well-defined services. Creating a task handler that encapsulates the task assignment and execution details is recommended. The service can then be exposed through a process control. In general, the activities performed for most of the task assignments are the same. Once assigned, the process waits for the task to be executed, aborted, or overdue. The process may then return the response back to the caller, and it also provides a way to handle exceptions in the common place in a consistent fashion. The task handler process, TaskManager.jpd, can be found in the sample code (available online at www.javapro.com).
- **Use the task creation XML document to create the task.** This document offers greater flexibility to create, assign, and associate data.
- **Assign overdue.** Many business cases may not require an overdue time, but to avoid stale tasks, set an overdue time that is well ahead in the future and notify the administrator of any overdue tasks.
- **Use the generic XmlObject for request and response documents.** The task manager is a reusable component. Instead of binding the input and output to specific schemas, use XmlObject to enable you to use it with any request and response document.
- **Dynamic routing using message-based routing.** WLI process is a convenient mechanism to route the document to a sequence of approvers. However, sometimes you may want to assign and reassign tasks dynamically when the routing is determined at run time based on certain parameters. This dynamic assignment can be achieved by keeping the list of approvers and order of the approvers in the document itself—a classic case of message-based routing—that enables you to change the approver or the sequence of approvers dynamically.
- **Use a response document to capture the user response.** Most of the approval scenarios require some kind of indicator to capture the user's response action. Some examples of user action are accept, reject, reassign, and hold. The out-of-the-box status for Worklist tasks are assign, claim, start, stop, abort, and complete. We recommend that you create a response schema that can hold the user-defined status properties like user response action and completed time. You will find the schema response.xsd that represents the user response in the sample code (available online at www.javapro.com).

ation to have the privilege of worklist administration. To make any user a worklist admin, edit the wliconfig/WorklistConfiguration.xml file in the domain to add the group of the user as worklist admin. By default, IntegrationAdministrator is added as worklist admin. Here's an example of the WorklistConfiguration file:

```
<WorklistConfiguration xmlns=
  "http://www.bea.com/wli/
  management/worklistconfig">
<taskTrackingLevel>basic
</taskTrackingLevel>
<taskCreationRole>Anonymous
</taskCreationRole>
<adminRole>IntegrationAdmin
</adminRole>
<adminRole>Admin</adminRole>
</WorklistConfiguration>
```

The task worker control is really a wrapper to the worklist manager EJB, which means that anything you do with the task worker control can be done by invoking the EJB directly from Java code. This EJB is useful when you want to manipulate the tasks from an external application or a POJO.

WLI's run-time database stores all the tasks that are not completed yet. Completed tasks are stored in the run-time database until the archiver is run. The archiver copies the completed tasks to the archive database. Reporting and audit may be run on the archive database. Depending on the purge schedule, archived data will be deleted from the run-time database. This archival process helps to keep the run-time database smaller and more efficient.

WLI Console provides an administrative interface for worklist administration and process-instance monitoring. You can assess the state of current business processes, identify relevant processes by performing comprehensive queries, troubleshoot problematic processes, monitor the progress of task completion against due dates, perform queries to show individual workloads, and even reassign tasks.

WebLogic Integration Worklist is a powerful feature to implement human workflow applications. We hope that the sample code, the example presented here, and best practices (see the sidebar, "Worklist Development Tips") provide a clear picture on building human workflows and enough information on how to leverage the out-of-the-box worklist example. JP

Anbarasu Krishnaswamy is a consulting technical manager with BEA professional services who has been working with BEA Systems for six years and BEA technology for more than 10 years. He is a Sun-certified Java programmer and BEA-certified WebLogic Server and Integration developer.

Vijay Mandava joined BEA as a technical manager in the professional services organization in 1999. He now works as a technical account manager in the systems engineering organization. Vijay is a Sun-certified Java programmer and a BEA-certified Weblogic Server developer. Contact Anbarasu at **anbarasu@bea.com**, and contact Vijay at **vmandava@bea.com**.

When Old Code Stops Working

Don't force upgrades with the latest and greatest development software. Give customers an upgrade road map instead

by Daniel F. **SAVARESE**

There is a camp of developers that has felt for a long time that Sun has been Java's worst enemy. From the tight grip Sun held over the platform and the resulting closed development model to numerous API shortcomings and even branding boondoggles, they had much to complain about. In the past two years, Sun has set almost everything right that many have felt it had done wrong.

Sun makes available weekly snapshots of the latest Java Development Kit (JDK) at its java.net portal. Licensing terms are more open than they have ever been. The Java Community Process (JCP) is more transparent and easier to participate in. Even the Java 2 branding has been replaced with something more sensible. If there ever was a real danger of Java developers jumping ship en masse, Sun has taken many positive steps to forestall that event. It may be too late to revive Solaris development, but Java development is only getting stronger.

Thinking about the way Java used to be managed has gotten me to thinking about old code. Recently, a programmer tried to convince me that I was doing the wrong thing by writing a class library that didn't depend on Java 5. I had chosen to remain compatible with JDK 1.2, but he was doing all of his development with the snapshot releases of JDK 6. The root of the disagreement was a difference in focus. If I were focused purely on my personal preferences as a developer, I would surely tie my code to the latest and greatest Java release. But I was focused on my customers' needs, which demanded that I provide compatibility with legacy software.

Reasons Galore

Customers often have good reasons to delay upgrading to newer versions of Java. Perhaps the most common reason is that old code may stop working. Bugs in a new Java Virtual Machine (JVM) may cause old code to stop working. This phenomenon was common during the first couple of years when just-in-time compilers (JITs) started to appear.

Changes in the Java language can befuddle old code. For example, the addition of new *assert* and *enum* keywords in the past two major Java releases have caused many programs to fail to compile. Changes in the implementation of JVMs can crash class files. For example, the JDK 1.0.2 compiler would inline final methods, sometimes generating bytecode where a private class member was accessed illegally. The JDK 1.0.2 JVM didn't generate any errors when executing such bytecode. The JDK 1.1 JVM implemented the Java specification more correctly and rejected such code.

The use of class file obfuscators also generated upgrade problems. Early JVMs were quite permissive about the symbols they would accept in bytecode. Later JVMs were more strict and would reject bytecode generated by obfuscators that relied on the use of invalid symbols. Early implementations of inner class compilation caused problems similar to those of the JDK 1.0.2 inline function access violations.

The addition of new classes, such as the MatchResult interface in Java 5 (which clashes with Jakarta ORO's MatchResult), can cause naming clashes that require you to change import statements or fully qualify class names. Adoption of new features before they've been fully debugged, such as the new I/O API in JDK 1.4, can cause you to wait for several maintenance releases before your code will work as expected.

I've covered only a handful of the situations where Java upgrades have caused problems in the past. There are any number of reasons why a customer must rely on an older Java environment. Only if you have both the source code to a program or library and the time to update them, can you circumvent legacy code problems. I'd like to take the opportunity to do a couple of updates of my own. Over the years, it has been brought to my attention that some of the code examples in my old *Java Pro* columns don't work. It's not that the examples didn't work when I wrote them.

Go Online

Visit www.javapro.com for related resources. Simply type the Locator+ code into the field in the upper-right corner of the page.

Download

JP0511 Download all the code for this issue.

Discuss

JPTALK Discuss this article in the "Talk to the Editors of *Java Pro* Magazine" discussion forum.

Read More

JP0511PS_T Read this article online.

JPEP030718EH_T Read the related article "Exploit XDoclet's Struts Capabilities" Erik Hatcher.

JP0303DS_T Read the related article "Get Small with Wireless Messaging and Mobile Media" by Daniel F. Savarese.

JP9901AP_T Read the related article "Step through AWT before You Swing" by Daniel F. Savarese.

They just stopped working as newer Java releases passed them by.

The question I get asked about most often pertains to an old question-and-answer item I responded to in my column, "Step through AWT before You Swing," in the January 1999 issue (see Resources online at www.javapro.com). At the time, the Swing API was still contained in com.sun.java.swing instead of javax.swing. A reader wanted to know how to place a JButton in a JTable cell and have the button respond properly when pressed. Apparently, the Q&A item is still published online independent of the column and pops up in Web searches from programmers who still have the same question.

Needless to say, the code no longer compiles because of the package name, but that's easy to fix. The problem people have with the code example is that it no longer works as advertised. At some point between JDK 1.1 and one of the J2SE releases, the

Listing 2 Wireless Messaging Revisited

```
@@ -25,8 +25,13 @@
  String getURL() { return urlField.getString(); }

  public void commandAction(
    Command c, Displayable d) {
-     if(c == start && d == this && openConnection(
-       getURL()))
-       setCurrentScreen(sendScreen);
+     if(c == start && d == this)
+       (new Thread() {
+         public void run() {
+           if(openConnection(getURL()))
+             setCurrentScreen(sendScreen);
+         }
+       }).start();
    }
  }
```

```
@@ -48,8 +53,12 @@
  public void commandAction(
    Command c, Displayable d) {
      if (d == this) {
        if(c == send) {
-         sendMessage(getMessage());
-         setString("");
+         (new Thread() {
+           public void run() {
+             sendMessage(getMessage());
+             setString("");
+           }
+         }).start();
        } else if(c == view)
          setCurrentScreen(receiveScreen);
      }
```

Blocking operations, such as network I/O, should be performed in a separate thread, or the emulator and actual device may hang.

Listing 3 Rewind to Mobile Media

```
@@ -9,40 +9,44 @@

public class MMAMidlet extends MIDlet implements
  PlayerListener {

-  class StartScreen extends Form implements
-    CommandListener {
+  class StartScreen extends Form implements
+    CommandListener, Runnable {
    private TextField urlField;
-    private Command start, stop;
-
+
    StartScreen(String title) {
      super(title);
      urlField = new TextField("url:", "", 128,
        TextField.URL);
      append(urlField);
-      start = new Command("Play", Command.SCREEN, 1);
-      stop = new Command("Stop", Command.SCREEN, 1);
      addCommand(start);
-      addCommand(stop);
      setCommandListener(this);
    }

    String getURL() { return urlField.getString(); }

+    public void run() {
+      playMedia(getURL());
+      urlField.setString("");
+    }
+
    public void commandAction(
      Command c, Displayable d) {
-      if(d  == this) {
-        if(c == start) {
-          playMedia(getURL());
-          urlField.setString("");
-        } else if(c == stop)
-          stopPlayer();
-      }
-      if(c == start && thread == null) {
```

```
+        thread = new Thread(this);
+        thread.start();
+      } else if(c == stop)
+        stopPlayer();
    }
  }

  private StartScreen startScreen;
+  private Command start, stop;
  private Player player;
+  private Thread thread;

  public MMAMidlet() {
+    start = new Command("Play", Command.SCREEN, 1);
+    stop = new Command("Stop", Command.SCREEN, 1);
    startScreen = new StartScreen("Media Player");
    player = null;
+    thread = null;
  }

  void playMedia(String url) {
@@ -56,7 +60,11 @@
    if(vc != null) {
      Item video =
        (Item)vc.initDisplayMode(
          vc.USE_GUI_PRIMITIVE, null);
-      startScreen.append(video);
+      Form form = new Form("Video");
+      form.addCommand(stop);
+      form.setCommandListener(startScreen);
+      form.append(video);
+      setCurrentScreen(form);
    }
    player.prefetch();
    player.start();
@@ -83,6 +91,7 @@
      startScreen.delete(startScreen.size() - 1);
    }
    player = null;
+    thread = null;
  }

  void reset() {
```

The mobile media example must perform blocking operations in a separate thread and display its video in a newly created Form.

implementation of JTable changed, and the technique I presented for using a button in a table cell stopped working.

Keeping Up

The technique I had presented was to implement your own TableCellRenderer and MouseListener. The TableCellRenderer would return the button as the cell renderer, and the MouseListener would forward to the button events that occurred in the cell. That approach doesn't work anymore. The MouseListener is no longer necessary, and you have to implement a custom TableCellEditor that behaves much like the custom TableCellRenderer. In addition, you must make the cell editable. Instead of reproducing all of the code for comparison, I have included a unified context diff (see Listing 1 online at www.javapro.com).

More recently, my article "Get Small with Wireless Messaging and Mobile Media," (*Java Pro*, March 2003) contained code that stopped working. I had decided to keep the code listings shorter by not using a separate thread for network I/O in the code examples. That worked fine on the original Wireless Toolkit (WTK) from Sun, but it probably didn't work reliably on real devices and no longer works on the WTK 2.2 release. If you perform a blocking I/O operation inside of a command handler, the device emulator (and likely the real device) will freeze up. Instead, you should spawn a separate thread as I crudely demonstrate in the wireless messaging example fix (see Listing 2). I should really declare a Thread variable for clarity and avoid using an anonymous class, but that's what I was trying to avoid in the original article to keep the code listing shorter.

In addition to the I/O issue, correct use of the emulator has changed. Instead of specifying datagram ports through the com.sun.midp.io.j2me.sms.DatagramPortIn and DatagramPortOut system properties, you use phone numbers defined by the wma.server.firstAssignedPhoneNumber and wma.client.phoneNumber properties. The default phone number used by the emulator is +5550000. Therefore, using port 2112 from the original example, you would use sms://+55500000:2112 and sms://5550001:2112 as the URLs to send SMS messages between two emulator instances.

That March 2003 article also contained a mobile media example that stopped working. The example suffered from the same blocking I/O problem as the wireless messaging example. Again, the fix is to perform the I/O in a separate thread (see Listing 3). Also, with WTK 2.2, the sound will play, but the video won't display. Instead of appending the video item to the already displayed start screen, the video item should be appended to an undisplayed form (see Listing 3) before setting the Form as the current screen.

Not only can code stop working because of development tool or run-time environment changes, but also it can stop working because of operating system upgrades. For example, the 1.1, 1.2, and 1.3 Linux JDKs all stopped working when I upgraded to a Linux distribution using glibc 2.3. There were some earlier upgrade issues involving thread models, but the glibc 2.3 upgrade permanently broke the JVMs from all of those JDKs and even the early JDK 1.4 releases.

The JDKs that broke all had these symbols in the libjava.so shared library:

```
U __libc_wait@@GLIBC_2.0
U __libc_waitpid@@GLIBC_2.0
```

Somehow, the library was linked against these private glibc symbols instead of the public wait and waitpid functions. The latest version of glibc stopped exporting those symbols, causing a run-time link error. You can't edit the libjava.so symbol table directly to use the public function because the offsets of the symbols in the rest of the library would get thrown off. Instead, you can create wrapper functions with symbol names of the same length (see Listing 4 online at www.javapro.com), put them in a shared library, and edit the libjava.so symbol table.

Even though it is a binary file, you can safely edit libjava.so with a Perl substitution:

```
perl -pi -e
  's/__libc_wait/mylibc_wait/g'
  libjava.so.
```

To make sure your shared library is loaded, you have to add it to the LD_PRELOAD environment variable, which you can place in the JDK scripts. That's enough to get JDK 1.3 working. For JDK 1.2, you also need to set the LD_ASSUME_KERNEL environment variable to 2.4.19 so the proper threading model will be used. In addition to that, for JDK 1.1.8, you need THREAD_FLAGS=green or LD_ASSUME_KERNEL=2.2.9 and THREAD_FLAGS=native.

Nothing Lasts Forever

Eventually, everyone has to upgrade. I have DOS programs I wrote and compiled when I was in high school that still work without changes on Microsoft Windows XP and Windows Server 2003. I don't expect every software environment to maintain that level of compatibility. However, I recently had to junk a perfectly good 486 PC because the CMOS battery was completely drained, rendering it unable to boot. The motherboard manufacturer had seen fit to use a charge accumulator instead of a replaceable on-board battery and provided no jumper pins and connector for an external battery. When the charge accumulator eventually lost its ability to recharge, the entire motherboard was useless. It's more cost-effective to throw away the computer than to spend the time tracking down the accumulator (which may be embedded in an ASIC), finding a replacement, and soldering it back onto the motherboard.

For enterprises that are stuck using software that lacks a replaceable on-board battery, it's cheaper to stay put instead of upgrading. Although I don't expect software to work unchanged forever, some consideration should be paid to customers. Satisfying their legacy software requirements within reason and preparing them for change with a smooth upgrade road map is far more considerate than forcing them to upgrade everything just because you like to use the latest and greatest development software. *JP*

Daniel F. Savarese is an independent software developer and technology advisor. He founded ORO Inc., was a senior scientist at Caltech's Center for Advanced Computing Research, and was vice president of software development at WebOS. Daniel wrote the original Jakarta ORO, Commons Net, RockSaw, and Sava Algorithms libraries. He also coauthored How to Build a Beowulf (MIT Press, 1999) and earned a Ph.D. in computer science from the University of Maryland College Park. Contact Daniel at **www.savarese.org/contact.html**.

The Year of AJAX

by Kito D. **MANN**

If your head wasn't in the sand during 2005, you've probably heard of AJAX (Asynchronous JavaScript and XML). Just in case you're wiping the sand out of your hair right now, here's a brief overview. AJAX is a rebranding of technologies introduced in Microsoft Internet Explorer (IE) several years ago. These technologies, branded by Microsoft as Dynamic HTML (DHTML) include HTML, an implementation of Cascading Style Sheets (CSS), plus the ability to manipulate a page with JavaScript through the browser's Document Object Model (DOM).

IE 5.0 quietly introduced something quite innovative (if proprietary): an ActiveX object called XmlHttpRequest. This object allows developers to write JavaScript code that communicates asynchronously back to the server, retrieves a response, and then dynamically updates the display *without requiring a full-page refresh*. Since updating part of a page is faster than updating all of it, applications appear much more responsive. (Note that you can dynamically manipulate the page with JavaScript even if you don't talk to the server first.) With AJAX, Web applications can behave more like their desktop counterparts using a Web browser (no special software or plug-ins are required).

Why is DHTML—I mean, AJAX—so popular *now*? There are a plethora of different opinions on this topic, but I think the answer is pretty simple. First of all, people are sick of the limitations of browser-based user interfaces, especially for intranet applications. At first, it was cool and hip to use new technologies and avoid deployment nightmares, but people have finally realized that it might be nice to have more than a few basic widgets to work with. Second, developers love Firefox. It's the first version of Mozilla to truly take off, so much so that Microsoft is releasing an *extra* update to IE just to compete. However, Mozilla has supported the XmlHttpRequest object since 1.0. In other words, AJAX is fully supported by all browsers (Apple Safari and Opera added support recently).

The immense interest in AJAX, since it was originally coined by Jesse James Garrett of Adaptive Path (www.adaptivepath.com) in February 2005, makes it probably the number one buzzword of 2005. So far this year we've seen several AJAX books, AJAX Web sites (I highly recommend www.ajaxian.com, run by Ben Galbraith and Dion Alamer, coauthors of the forthcoming book *Pragmatic Ajax* [Pragmatic Bookshelf, 2006]), and of course an endless supply of AJAX frameworks.

Why do we need AJAX frameworks? As it turns out, once everyone started writing a bunch of JavaScript again, they remembered how much of a pain it can be, especially since there aren't too many good tools on the market. While AJAX is a powerful technology, it's pretty low level, and anything that's low level could use a few layers of abstraction (even Microsoft has a framework called ASP.Net Atlas).

What does this have to do with Java? A lot, because people use Java to build quite a few Web applications, and, as with any new (or newly rediscovered) technology, the industry is still trying to figure out how to use it best. People are still figuring out how to handle accessibility and how to avoid excessive bandwidth usage. Some people think that if you're using AJAX, you no longer need a Java Web application framework (theoretically, the browser can talk directly to Web services!). Others have tried building a lot of custom AJAX code only to quickly realize why they haven't written any JavaScript in a long time.

The Web application framework market has been quick to respond; you can now use AJAX with JavaServer Faces (JSF), Struts, WebWork, Tapestry, and so on. These frameworks handle things in different ways, but the key point is that it's possible to leverage what you already know. There's no need to reinvent the wheel. And unless you're anxious to manage all of the complexities of the browser DOM, JavaScript, and CSS, you probably shouldn't.

The bottom line is that AJAX is here to stay, and it's great to see that the Web has quickly realized a powerful, (sometimes) backwards-compatible evolutionary path. Fortunately, the Java ecosystem is dynamic enough to rapidly support new technologies, and you can use your existing Web framework with different AJAX frameworks.

Component-based Web frameworks such as JSF give you the best of both worlds: components that have both server-side and client-side representations, which means that developers are shielded from the complexities of AJAX and use the same event-oriented programming model whether the components are displaying AJAX code; WML; XUL; or plain vanilla HTML.

AJAX is a powerful way to use a set of ubiquitous technologies, and it's here to stay (although, I'm not convinced the buzzword will be immortal). For Java developers, architects, and managers, the key is to make the best decision for the project at hand. More often than not doing so means using frameworks, either on the client, the server, or, better yet, both. The most important rule, however, is to remember that any technology should be used only where appropriate: every single Web application doesn't need to be "ajaxified," and those that can benefit from AJAX may be fine with just a few "ajaxian" features. JP

Kito D. Mann is editor-in-chief of JSF Central (www.jsfcentral.com), principal consultant at Virtua Inc., and author of *JavaServer Faces in Action* (Manning, 2005).

FTPOnline

One Source for All Your Technical Information

Newly Expanded, Easily Accessible

1 CHANNELS
To better serve your needs, FTPOnline has been restructured around seven channels: Architecture, Java, .NET Development, Windows IT, ASP.NET, Database and Security. More channels to come!

2 SPECIAL REPORTS
Get comprehensive information on subjects critical to all IT professionals, such as Security, Service-Oriented Architecture, and Operations Management.

3 WEBCASTS
Watch and listen to industry experts discuss hot IT topics.

4 WHITE PAPERS
Download white papers that examine evolving technologies.

5 RSS FEED
Get quick updates on the latest blogs and articles published at FTPOnline.

6 MAGAZINES
Filled with downloadable code, interviews with industry visionaries, in-depth tutorials, overviews of implementation and management strategies, article archives, and more!

7 NEWSLETTERS
Free e-mail newsletters in your area of interest, delivered right to your inbox.

Go there today:

www.ftponline.com

FTPOnline

Sybase® WorkSpace: Bridging the Gap Between SOA and Traditional Tooling

Sybase WorkSpace, a unified application development environment, is the first designed to bridge the gap between the vision of a service-oriented architecture (SOA) and the reality of what traditional development tools can deliver. Sybase WorkSpace combines modeling, data management, services assembly and orchestration, Java™ development and mobilization in a single tool. Designed to support the principles of service-oriented development of applications (SODA), Sybase WorkSpace enables developers to quickly build and deliver many kinds of applications: from event- and data-driven applications to web-based, composite, and mobile applications. Sybase WorkSpace is built upon the Eclipse open source framework, making it easier and faster for developers to build complex applications that leverage heterogeneous infrastructures.

Get Sybase WorkSpace and start building tomorrow's applications today.

For more information and to download white papers, visit www.sybase.com/workspace

SYBASE®