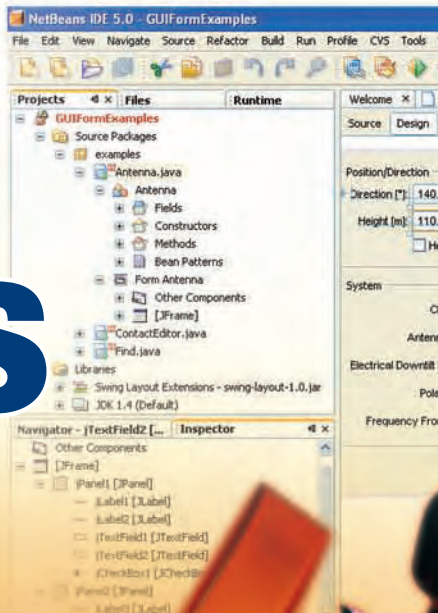
Borland Enhances Middleware for SOA, Java EE

# JAVAPro™

www.javapro.com A Fawcette Technical Publication 2006 Vol. 10 No. 2

# The Power Behind IDEs

## Take a Fresh Look at Today's Tools




Clean Up Complex Java Code Bases

When Lazy Programming Is a Good Approach

Java ME: The "Write Once, Run Anywhere" Platform

Apply RCP to Enrich Enterprise Applications

Find Innovations Galore in Java Standardization

Rational®

_INFRASTRUCTURE LOG

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

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

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

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

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

JAVA Pro™

## COVER STORY



## FEATURES



## COLUMNS



## DEPARTMENTS



***Java Pro*** (ISSN: 1096-4495) is published quarterly 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
Daniel F. Savarese, and Peter Varhol

ART & PRODUCTION
VICE PRESIDENT, ART & PRODUCTION, Michael Hollister
SENIOR ART DIRECTOR, Bruce Gardner
PRODUCTION MANAGER, Kathleen Sweeney Cygnarowicz
SENIOR INTERACTIVE ART DIRECTOR/WEB PRODUCER, Lyndon Lloyd
ASSOCIATE WEB PRODUCER, Shane Lee

ADVERTISING SALES
AD DIRECTOR, VSM AND JAVA PRO, Kevin White
EXECUTIVE ASSISTANT TO THE VICE PRESIDENT, PUBLISHING, Susan LaCroix

CIRCULATION
SENIOR CIRCULATION DIRECTOR, Karen Koenen
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
MARKETING EDITORIAL PLANNER, Katie McGillivray
EXHIBIT SALES MANAGER, Tina Fontenot

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
ADVERTISING DIRECTOR, Roy Kops
PROJECT MANAGER, Fred Perry
SENIOR EDITOR, Terrence O'Donnell
ASSOCIATE EDITOR, Lauren Dresnick
EASTERN REGIONAL SALES MANAGER, Dennis Leavey
WESTERN REGIONAL SALES MANAGER, Lisa Sidlow

FAWCETTE TECHNICAL PUBLICATIONS
PRESIDENT, James E. Fawcette
VICE PRESIDENT, CHIEF INFORMATION OFFICER, Aaron Weule
VICE PRESIDENT, PUBLISHING, Jeff Hadfield
VICE PRESIDENT, CONFERENCES, Tim Smith
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, Terrence O'Donnell, *Java Pro* editor, and other FTP editors sounding off on IT and development issues.
**www.ftponline.com/weblogger/**

## H-1B Programs and Doctoral Imports Misguided

**Locator+ code: JF_060407**

**http://www.ftponline.com/weblogger/forum.aspx?ID=1&DATE=04/07/2006#586**

*by Jim Fawcette*

Unemployment in IT remains higher than the national average, and escalating pay for top-tier experts distorts average pay for programmers, Jim Fawcette noted in a recent blog. Find out more about his thoughts on this perspective of H-1B visa programs and the undermining of America's competitive position because of educational policy.

## More Online Exclusives

### Special Report: Mobile Java Development

**Locator+ code: MOBLJAVA**

The mobile industry is evolving rapidly, with certain platforms, tools, and languages beginning to establish footholds that are giving developers more standardized alternatives to realize full wireless functionality in their applications. Take a look at the state of mobile Java development from a variety of angles, including the Java ME platform APIs, a roundtable discussion of industry experts, and the latest versions of the Symbian 9 OS and Nokia's Series 40 3rd Edition and S60 3rd Edition platforms.

### Combat Increasing IT Complexity

**Locator+ Code: EA0601FB_T**

*by Firdaus Bhathena*

Given the challenges of complexity, how can you get an accurate picture of the systems, applications, and other infrastructure components in your environment? Consider some suggestions for exploring how companies can conquer issues associated with IT complexity and achieve success in their architecture initiatives.

**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.

Locator+ Code: jf_040510 go


**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 Java EE, middleware, and application servers; Java ME, devices, and embedded development; data access; servlets and JSP; Web services; and/or XML. Sign up for *free* at www.javapro.com.

# Java Season

by Terrence **O'DONNELL**

The JavaOne Conference brochure's tag line is "The Power of Java," and it certainly seems fitting given the healthy state of everything that is Java. The agenda for this annual, Spring season conference exemplifies how much innovation is out there and how much there is to learn to leverage Java technologies and build even more impressive enterprise-scale applications.

Consider the breadth of topics for the platform tracks alone. Presentations for Java SE highlight the platform's version 6 ("Mustang"); Web 2.0, NetBeans, robust Java technology-based applications, and of course the ballyhooed Eclipse Rich Client Platform; and other noteworthy technologies like JMX, Swing, Struts, and REST. As noted in the brochure, some Java SE topics overlap topics geared for the Java EE platform track, and there is some instructive fare tied to this platform too. Not only will you find presentations for the de rigueur technologies like SOA, EJB, BPEL, and Java-.Net interoperability, but you'll find a variety of perspectives on ease of development, AJAX, and JSF. Of course, there is plenty of content on Java Specification Request (JSR) updates across all platforms, and in this issue's Public Static column Onno Kluyt, chair of the Java Community Process (JCP), provides a comprehensive summary of what's current with specific JSRs and maps them to their corresponding JavaOne sessions.

Arguably some of the most exciting Java-related innovation is occurring in mobile applications development, and this vitality is reflected in some noteworthy JavaOne session offerings around new Web 2.0 services, Blu-ray, the NetBeans Mobility Pack, and UI design for Nokia's Series 40, S60, and Series 80 platforms. In fact, innovation for the mobility market is heating up appreciably in the North America region, and for some good insight into this trend I urge you to visit the FTPOnline site to see our special report on mobile Java development (www.ftponline.com/special/mobilejava/), which includes articles and resources that reflect the dynamic state of the mobile space.

To give you some report highlights, we gathered industry insiders from organizations that are members of the Forum Nokia Pro program for a roundtable discussion of enterprise application development for mobile devices. For an update on Java ME specifically, Michael Yuan, a recognized expert on end-to-end mobile and enterprise solutions, provides a detailed discussion of the platform and its latest advancements (that article also appears in this issue). Periodic *Java Pro* contributor Rick Grehan offers a clever technique for reverse engineering a typical mobile application to leverage an object database library for UIQ-based devices. Also, the report includes technical articles on applications development for the MIDP stack for Symbian-based mobile devices and Nokia's Series 40 3rd Edition and S60 3rd Edition platforms. For some additional insight you'll find an exclusive interview with Lee Epting, vice president, Forum Nokia, who discusses the rapid progress Forum Nokia and Forum Nokia Pro are making to support innovation for developers of mobile applications.

Tools constitute another JavaOne track, and Java-based tools continue to be an area exhibiting a lot of creativity to make the development process easier for developers and developer teams who find themselves constricted by tight budgets and more compressed time-to-market milestones in their production environments. Since productivity is a key criterion for tools evaluation, our cover story by frequent contributor and columnist Peter Varhol analyzes the value and productivity proposition available from a select group of IDEs, which are freely available for evaluation.

JavaOne's importance to the industry cannot be overstated, and the publisher of this magazine intends to participate with two key events. Our Java Technology Roundtable returns—after last year's hiatus—to bring together industry luminaries who will share their perspectives on current technology trends for Java in the enterprise, what has transpired in the Java development space over the last year, and where the industry is going in the year ahead. Look for expansive coverage of the roundtable both online at FTPOnline and in an upcoming issue of *Java Pro*.

We also host our annual Java Technology Achievement Awards during JavaOne, where we present awards for outstanding Java-based products selected by your votes. Readers responded to an online ballot to vote for their favorite tools and technologies among 20 categories, and we are pleased to recognize the winners, including two additional community awards selected by *Java Pro* editors. Like the Java Technology Roundtable, look for exclusive coverage of the award winners both online and in an upcoming issue of this magazine.

A lot of exciting opportunities for enterprise- and Java-based development are now available to the community. Look for an upcoming special report on FTPOnline that will spotlight the current state of Java, Eclipse tools, and JavaOne highlights. As always, if you have suggestions for our coverage, send me an e-mail. JP

Terrence O'Donnell, Editor
todonnell@fawcette.com

# Using Ajax?
# Log in and Learn from the Experts


NEW!

**FTPOnline's Virtual Tradeshow on Ajax**

**August 15 & 16, 2006**
**www.ftponline.com/ajax**

**Visit** www.ftponline.com/ajax to register for FTPOnline's premiere Virtual Tradeshow.

**Access** the **Free** in-depth tutorial on Asynchronous JavaScript and XML (Ajax) from your desktop.

## Benefits:

- Two full days of virtual sessions and keynotes led by Ajax experts
- Examples of real-time form data validation and server-side notifications
- Best practices for creating interactive Web applications
- Chat with virtual exhibitors about your Ajax concerns

For sponsorship opportunities, contact Roy Kops at 650-378-7144 or rkops@fawcette.com.

FTP FAWCETTE TECHNICAL PUBLICATIONS

# Mission-Critical Optimization

## Borland VisiBroker 7.0 enhances SOA support and adds capabilities for more control over CORBA-based applications

Borland Software Corporation announced recently a significant new release of its VisiBroker product. Visibroker 7.0 is an enterprise middleware layer that is optimized for mission-critical applications and provides the ability to expose as services CORBA application functions in service-oriented architectures (SOAs). Concurrently, Borland also announced the release of the 6.6 version of its Borland AppServer, a high-level, J2EE application server that supports the J2EE 1.4 standard and can be embedded into applications and environments that employ tight integration among CORBA-J2EE applications.

VisiBroker has a long history at Borland that began with a ten-year-old acquisition of Visigenics, which at that time was a supplier of a middleware product that was based on the CORBA standard. Though CORBA's prominence as a standard for building enterprise-class, distributed applications that required the reliability and scalability necessary for mission-critical applications gave way in the late nineties to Java and the J2EE platform, significant deployment of CORBA between the late 1980s to the mid 1990s means that many substantial "pockets in the world where CORBA is still a very viable and heavily deployed technology" remain, according to Raj Sehgal, senior director of product marketing at Borland.

"At a fundamental level CORBA is just a standard definition of how you build applications to talk to each other in a very tightly coupled way," Sehgal said. Because today's applications have a lot of complex, transactional data going back and forth, requiring a high degree of reliability, scalability, and synchronization, the integrity and the synchronization of the data needs to be guaranteed. CORBA, Sehgal said, is still very well suited for this role, even in today's enterprise environments.

Borland's middleware products parallel the company's application development tools and application life-cycle management (ALM) lines of business. However, the middleware product line, of which the lead product is VisiBroker, is deployed in a run-time production environment; the middleware is embedded in applications and invoked when those applications are run. In addition to VisiBroker, Borland's J2EE-based AppServer builds on the VisiBroker technology along with other ancillary products that support interoperability across multiple, different applications.

"For the middleware market, even though at a macro level CORBA is flat to maybe slightly declining, it's still, we believe, an over $200–250 million business worldwide," Sehgal said. "Borland has a market-leading share of that [space] in terms of licensing revenue. The J2EE application server is a well-tracked market, and obviously there are some big players like IBM, BEA, Oracle, and others. What we have is a smaller, but very healthy business with our J2EE application server, mostly on the high end, that follows the lead of the VisiBroker product line, and customers come to us for the J2EE business because they've been using the CORBA-based product."

Prior to these announcements, Borland was shipping the 6.5 versions of both products. While the 7.0 version is a major new release for the VisiBroker line, Borland AppServer has undergone a minor upgrade with its 6.6 release; however, Sehgal said there will be a major release of the J2EE application server some time in 2007.

Organizations can use VisiBroker 7.0 to integrate distributed applications that may be built using various languages, platforms, and standards. Enterprise data "locked" in older CORBA applications can be leveraged into newer applications that are built on other technology stacks such as Web services, Microsoft .Net, or Java EE. Applications native to the .Net platform can participate equally in a set of tightly coupled cooperating applications with CORBA applications written in C++ and Java. VisiBroker also supports MontaVista Software's carrier-grade Linux operating system platform for telecommunications and data communications.

For more information about both middleware offerings and pricing, visit Borland's Web site.

**Borland Software Corporation**
**800-632-2864; 408-863-2800**
**www.borland.com**

**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.



**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.

**Do you really want to write Ad Hoc Build and ALM Scripts on a nice day like this?**

## Stop scripting. Use Openmake® and enjoy your summer.

Openmake 6.4 accelerates the Build to Release Workflow using a 100% reusable framework. It minimizes ad hoc scripting so you can go home and enjoy your summer.

- Configure builds for Eclipse, .Net, Visual Studio, IBM-RAD, Java, J2EE, SOA and enjoy the benefits of reusable builds.
- Manage your ALM Workflow after the Build from the Execution of Test Scripts, Email Notifications and Deployment.
- Schedule Builds and ALM Activities
- Run Build and ALM metrics Reports.
- Use SCM Inspectors for viewing Difference Reports and Item History
- Create Footprints for meeting tough IT Goveranance Standards

openmake® a product of catalyst SYSTEMS CORPORATION

Optimized for Microsoft Visual Studio.net

eclipse READY

Ready for IBM Rational. software

**www.openmake.com** Call 800.359.8049 to schedule a web-based demo.

# Finding the Best Value in Java IDEs

**In an era of open source and free tools, cost and developer productivity weigh heavily on making an IDE choice**

by Peter **VARHOL**

Java developers and developer teams have many alternatives when choosing among commercial development environments. In the past, most made this decision by a combination of familiarity, cost, and technical applicability to the project at hand. Because many IDEs in the past had similar feature sets and costs, often the preference was based on familiarity.

**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**

**JP0602** Download all the code for this issue.

**Read More**

**JP0602PV_T** Read this article online.

**JP0509CS_T** Read the related article "Adversaries and Partners" by Chris Schalk.

**JP0506MM_T** Read the related article "Eclipse Unleashed" by Mike Milinkovich.

**JP0311PR_T** Read the related article "WebLogic Platform 8.1" by Daniel F. Savarese.

However, these decision factors don't fully define the value that is delivered by the selection. *Value* in this case doesn't mean the number of features or ease of using those features, but rather by productivity in performing specific common tasks that developers do daily, and the cost of that productivity in terms of the product and support cost. As the cost of tools continues to drop, and there is less differentiation between core features, value and productivity are coming to the forefront as key differentiators in tool selection and use.

The equation has been complicated by the availability of free and open source integrated development environments (IDEs) such as Eclipse and NetBeans. At first glance, the best value might seem to be delivered by the free solutions. However, free doesn't necessarily translate into productive. It is likely that the most expensive part of the development process is the developer's salary, so optimizing the use of time is a key consideration.

The question of productivity is a big one, and few have attempted to even consider productivity when evaluating development tools. It is difficult to measure in an absolute sense because of both the nature of developer skills and the nature of the projects to which those skills are applied.

The requirements for productivity are real and demonstrable. Productivity means being able to do a specific set of activities faster or more efficiently in one way over another. The goal is to save development time and effort, which is typically the most expensive part of the development process. The investment in tools must significantly raise productivity, yet not cost so much as to diminish the value of that increased productivity.

This review looks at the features provided by the IDE, what they add to the ability to build common applications, and what a developer would have to do without them. As such, the products examined here represent different models of value and developer productivity: Eclipse-based versus proprietary, one-time license versus subscription, and so on. The intent is to look at a combination of cost over time and features to derive the maximum value.

With these criteria in mind, we'll look at Borland JBuilder, IBM Rational Software Architect, BEA WebLogic Workshop, Oracle JDeveloper, and Genuitec's MyEclipse Enterprise Workbench. The products we're focusing on here can be downloaded for evaluation and purchase, rather than requiring retail purchase, shipment in a box, or interaction with a sales representative (in most cases downloading requires registration or membership, but in all cases evaluation was free). While this method of procurement was convenient for purposes of review and analysis, it also represents the way a lot of developers prefer to obtain new tools.

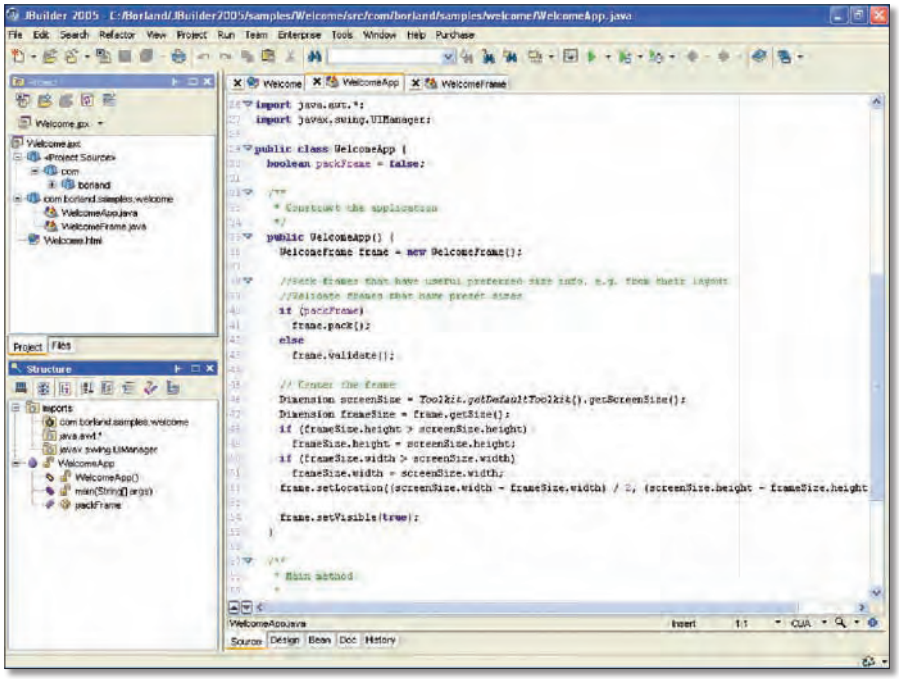


**Figure 1 | Feature Rich** Borland JBuilder's integrated features incorporate refactoring, a slick editing environment, Javadoc support, code profiling, JSF, Struts, and Web services designers.

The test machine for this analysis was a Dell notebook with a 1.6 MHz Centrino processor and 512 MB of memory. While this machine isn't an especially powerful computer, it is probably representative of the average of development computers in many enterprises. All of these products were applied to build a specific application: a simple Web-based time and attendance system. It enables workers to log on and time-stamp their start and end dates, times, and calculate wages based on hours worked. The application includes several user interface pages, a simple calculation engine, and a back-end MySQL database. While the architecture and coding of this sample are relatively simple, it is likely representative of many applications that are developed for custom enterprise use.

## Borland JBuilder

Borland JBuilder 2005 is the last version of this signature product that remains on the proprietary platform; future versions will be built on Eclipse. However, today the current JBuilder comes in three versions:

- Personal Edition – This version is freely downloadable and provides the fundamental IDE tools plus a few additional extras such as a GUI designer, JUnit framework, and some other utilities.
- Developer Edition – This version adds a host of features, especially XML and Web support, and the latter includes JavaServer Pages (JSP) and JavaServer Faces (JSF).
- Enterprise Edition – This version adds Web services, Java EE and CORBA support, and UML diagramming.

The Developer Edition was used for testing purposes in this review. Although this edition is reasonably priced at $499 and provides a good feature set, it lacks the ability to build many common Java EE applications. You can use it to build

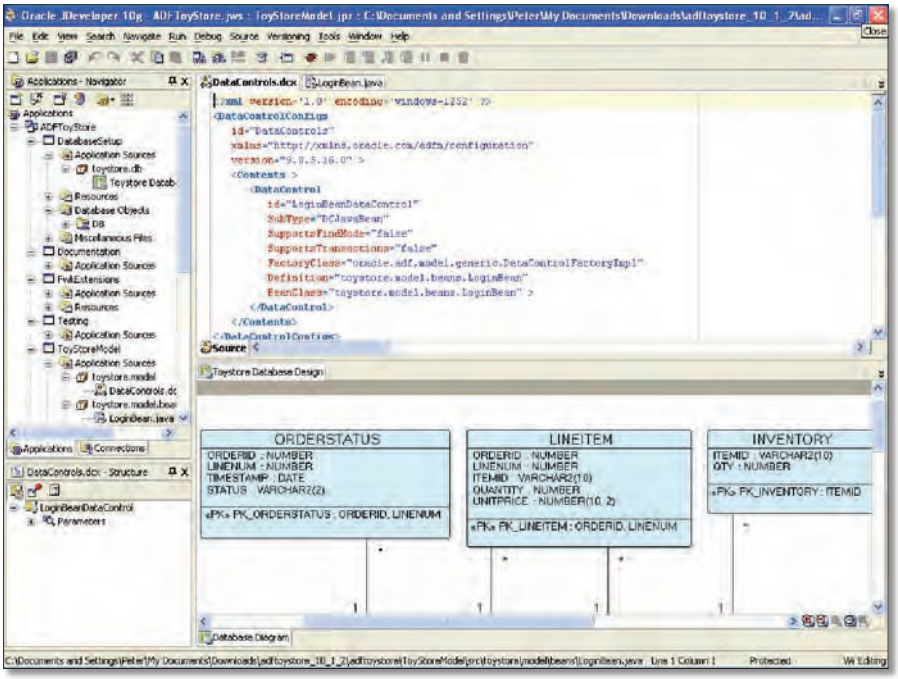



**Figure 2 | Single Environment** Oracle JDeveloper's unique development environment makes it possible to do design, development, and fine-tuning in a single location.

such distributed applications, but only by adding open source tools such as JBoss. In addition to the package as it stands, Tomcat was downloaded for a servlet engine to build the application as it was designed.

Despite this limitation, JBuilder is an enjoyable product to use. Its maturity (it is the only product reviewed here that has remained fundamentally the same for the last eight years) means that Borland has had the time to fine-tune the user interface to ensure a smooth developer experience.

JBuilder is a highly refined product with a number of integrated features. It incorporates refactoring, nice editing features, Javadoc support, code profiling (from the Borland OptimizeIt product), JSF, Struts, and Web services designers that speed the development process and improve developer productivity in those areas (see Figure 1). However, UML modeling is offered only in the Enterprise Edition, and only with two types of diagrams (other diagrams are available with Borland's high-end Together modeling product).

One other advantage with JBuilder is that as a fully Java-compliant development environment, it offers versions that run on Windows, Linux, or Solaris. Performance has been tuned over the years, but it still feels slow in launching and selecting features.

Where JBuilder might have a disadvantage is in its very plethora of features. Because it is a mature IDE, it has had a number of versions with incremental new features and capabilities. For example, this version includes enhancements to editing and a global gutter for tracking errors and opening the files associated with those errors. It also has good support for XML and support for the Java ME platform and WAP. However, while JBuilder offers modeling in the next version up (the Enterprise Edition), the company that bought one-time leader Togethersoft has not at all expanded that product offering, with only two UML diagrams available.

Because JBuilder's future road map is uncertain, it is difficult to extrapolate the advantages of the current platform into the future. It is likely that a future Eclipse version will, out of necessity, be less functional and less well integrated than today's product, and the $499 price means that it has to have a clear advantage over free and low-cost solutions to enjoy significant productivity advantages. The Enterprise Edition, at a full $3,500, adds the ability to create Java EE applications and testing tools, but the productivity payback may well be harder at that much higher price point.

## IBM Rational Software Architect

IBM Rational Software Architect 6.0 is one of the products examined here that is based on the Eclipse framework. In addition to the Eclipse Foundation software, the IBM product includes UML modeling, the full WebSphere Web server, other IBM tools for developing portals, and tools for identifying and refining patterns. It has a lot of software tools for a single user IDE; some users will appreciate the wide range of included tools, but others will find that they make the environment unnecessarily complex.

The modeling tools support nine UML diagrams, a remarkably complete solution. Since IBM can leverage the traditional Rational modeling tools, it is no surprise that Rational Software Architect has the best modeling solution. It is possible to create complex software models and generate at least some of the code required by the modeled application.

When combined with IBM's rule-based code analysis, these tools help an architect see how well projects are being implemented and how they fit within design guidelines and site requirements. In addition to structural and object-oriented patterns, Rational Software Architect can recognize and analyze seven of the Gang of Four design patterns. While that is a small subset of the full set of design patterns, it represents the only attempt among the tools here to support formal design patterns. This support offers the unique capability of ensuring that authorized patterns are followed during development.

Rational Software Architect has some support for C/C++ development, in addition to full support for Java. The modeling tools can perform transformations to C++, and various source code tools can analyze C++. However, the C++ IDE lacks a compiler and debugger, which must be obtained and installed separately. You can install your own if you already have one of these tools as an Eclipse plug-in, or you can download the GNU C++ compilers to do this. These features seem like an odd and incomplete addition.

The problem with Rational Software Architect is that it has the feel of a product that is really an amalgamation of several distinctly different tools. While the feature set is pretty complete, the major

# FTPOnline

# One Source for All Your Technical Information

## Newly Expanded, Easily Accessible

**1**

### CHANNELS
To better serve your needs, FTPOnline has been restructured around eight channels: Architecture, Business, 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 Mobile Java Development, SQL Server, and Application Lifecycle 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

pieces of the environment were all separate products at one time, and the differences show. Another limitation is that it tends to be slower than Eclipse by itself, probably because the memory on the test system was insufficient to contain a reasonable working set when using a number of the tools.

At $5,500, there is a lot of capability in Rational Software Architect, almost certainly more than most developers can use. The biggest issue is likely to be that developers will believe they are paying a premium for tools they don't need. While the product may have many tools that individually can improve productivity, the aggregate cost may be too high for many developers.

## BEA WebLogic Workshop

BEA is better known as a Web services vendor, but the company also provides a fine development environment with its WebLogic Workshop. BEA WebLogic Workshop 8.1 is a Java development environment that enables IT to visually build and assemble enterprise-scale Web applications, Web services, JSPs, portals, and Enterprise JavaBeans (EJB) for a service-oriented architecture (SOA).

WebLogic Workshop is a highly mature product with many features and a fine feel. Most longtime developers will feel very comfortable working in this environment. In particular, building EJBs or even Web services seems like a straightforward process, although neither were a direct part of this testing.

In the test application, code was written quickly, although it was less adept at allowing communication with the back-end database. And because EJBs or Web services weren't being used, many of the enterprise features weren't used. BEA has a small community of partners developing extensions to WebLogic Workshop; however, all of the partners are commercial entities that require separate purchase and maintenance agreements. These partner offerings also tend to be entire products, rather than simple single-purpose tools.

BEA also has a newer, Eclipse-based development solution. BEA Workshop Studio includes sophisticated WYSIWYG editors and BEA's AppXRay technology, which provides a view of the Web application as a whole. XRay helps provide depth and capabilities in code completion, consistency checking with generated classes, configuration files or annotations, prebuild error checking, and validation. The latest release includes annotation-driven EJB tools and bundles the Spring IDE Project for Spring Bean development.

The packaging and utility of this alternative makes the future of the original WebLogic Workshop somewhat doubtful, even though there is a beta of the next major release available. Nevertheless, WebLogic Workshop is an excellent supplement to the WebLogic application server. It is still capable outside of that deployment architecture, just not as well.

For development, WebLogic Workshop's price is certainly right, as it is freely available for use in development. If you are doing enterprise development with EJBs, and especially if you are deploying on the WebLogic application server or portal, Workshop is a natural choice and probably your most productive alternative. For other deployment platforms and for smaller projects its enterprise features can be confusing and unnecessary.

## Oracle JDeveloper

The JDeveloper IDE integrates all of the features needed by a developer building a Java application. Unlike some other Java development environments, with JDeveloper it's possible to move from design through development and tuning without leaving the environment. When you initiate a project, you can begin with a UML model. You build the model in two parts: build the activity diagram to define the behavior of an application, while laying out the structure of the application using class diagrams. Although this isn't a complete UML model by any means, it's enough to generate both class definitions and a state transition in code, which makes it useful for initial design (see Figure 2). However, it's not as seamless as it could be when moving into code.

The JDeveloper integrated code profiler is a useful debugging tool for most applications. It profiles application execution, memory utilization, and event sequence. In addition, the debugger works locally, remotely, or across multiple processes. You can also use the integrated CodeCoach to provide hints to improve performance or the use of Java technologies in your code. The principal limitation of JDeveloper lies in its UML modeling, where the product supports only four diagram types: activity, class, sequence, and use case. That number is usually enough to get you started, but

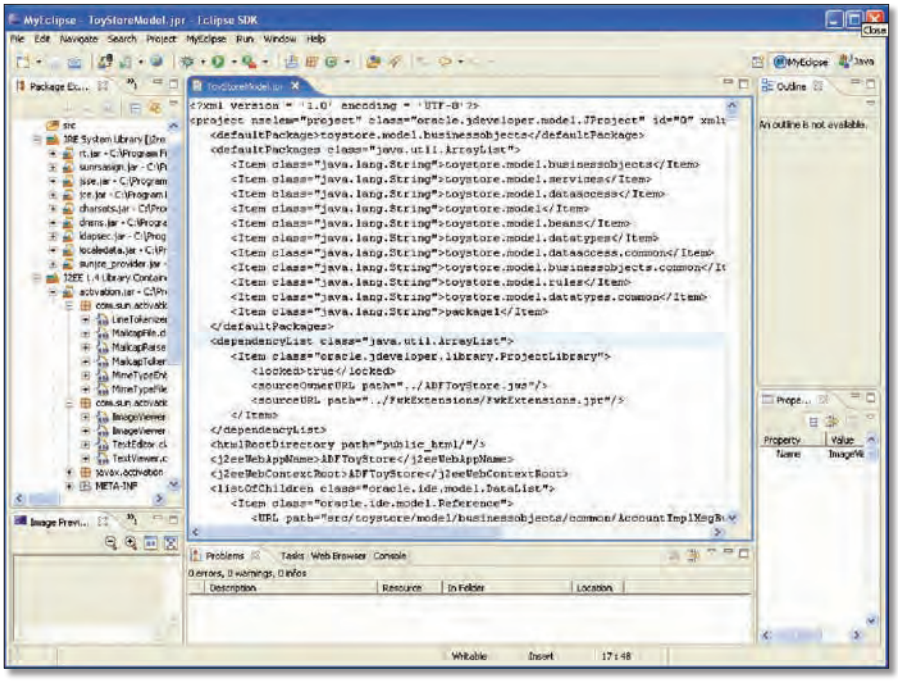



**Figure 3 | Tool Bonanza** Utilizing the Eclipse platform, MyEclipse provides an environment that integrates selected open source development tools and adds features based on them.

many UML practitioners also like using component and deployment diagrams for packaging and distribution.

However, JDeveloper also had some significant limitations. It consistently got the poorest marks in several categories, including compiler/interpreter performance, editor, libraries and frameworks, and the ability to integrate third-party tools. For developers working on a variety of projects, it lacks a number of tools, and Oracle has declined to form the third-party community that Eclipse, and to a lesser extent Borland, have fostered; there are few add-ins and limited ability to add additional tools.

An important feature in JDeveloper is the ability to use a set of libraries called the Business Components for Java (BC4J). BC4J uses Java database connectivity (JDBC) to provide an object-relational mapping of information stored in the Oracle9*i* database. It allows business logic to be centralized at the middle tier, as you would do when using a servlet, while leaving presentation-related activities to the JSP.

JDeveloper has integrated source control through the Oracle Software Configuration Manager. It supports an API for third-party source control packages, such as ClearCase and the open source CVS. It also incorporates support for working with hosted files on any WebDAV-enabled server.

Clearly, even though some of the features, such as UIX and BC4J, are used only with other Oracle products, there is still value here for the average Java EE developer. However, the question is whether the remaining features can make a developer more productive. For those developing Java EE, JSP, XML, or servlets in conjunction with Oracle database tools, JDeveloper is undoubtedly the toolset of choice.

Outside of the Oracle world most of these features did little good. With the MySQL database, or any third-party database for that matter, you have to regress to using standard techniques for database access and partitioning into layers, rather than BC4J. Some features of the UML models also assume that the target is an Oracle database.

## What About NetBeans?

As you download, install, and use NetBeans (currently at version 5.0), you can't help but think that in an alternative universe, NetBeans would hold the exalted role of open source flash point for the developer community currently occupied by Eclipse. NetBeans has many of the same features as Eclipse, yet arguably in a more productive package. And while Eclipse has broadened from its roots as an IDE to that of an all-encompassing, life-cycle platform and even application framework, NetBeans has remained focused as a tool for the developer.

Certainly there is nothing about NetBeans that might have prevented it from achieving such a role. Its significant features include a useful list of code refactorings, extensive code completion algorithms, and integrated CVS support for team development. It incorporates Ant as its build utility and project metadata repository, which makes it possible to export projects to other IDEs, which can then load a project developed in NetBeans and make edits and builds.

Probably the best feature of NetBeans is the Matisse forms designer (see Figure 4). Matisse lets you build a Swing-based form in a drag-and-drop manner, similar to other form builders in any language. However, Matisse lets you line up controls on a form far more easily than comparable tools, and the designed form also looks much more like the real thing. Matisse is so good that Genuitec recently announced a port to the Eclipse platform for its MyEclipse IDE.

NetBeans includes three optional downloads: a code profiler, a mobility pack, and a platform. For this discussion, only the profiler was looked at. It is surprising that more developers don't write a code profiler, something the Java Virtual Machine makes possible through profiling hooks. The NetBeans profiler is fun to use, and it is instructive to see how much you can sometimes change performance by changing a few lines of code. The mobility pack lets you build applications for embedded devices, while the NetBeans platform is a framework for building targeted applications.

As an IDE, NetBeans is at least comparable to Eclipse or any of the other alternatives, yet it barely gets a mention in the same breath. You can argue that Sun Microsystems handled the product poorly, or that it didn't have the exposure or marketing resources of the IBM-backed Eclipse, but you cannot easily argue that it is technically inferior. While it lacks the community and developer enthusiasm of Eclipse (it does have plug-in developers, but the number looks to be easily an order of magnitude smaller than that of Eclipse), technically it is a fine product that any development team can easily adopt and put to productive use. And if the Eclipse community ever looks toward the next new thing in Java development, NetBeans might just get to experience that alternative universe.

## Genuitec MyEclipse

MyEclipse is unique in this review, in that it includes few if any features that are developed using the traditional commercial development model. Instead, MyEclipse takes the Eclipse platform and integrates a number of other open source development tools into the environment, as well as adding features based on those tools (see Figure 3).

MyEclipse can be downloaded from the MyEclipseIDE.com Web site. It requires the previous download and installation of the Eclipse platform from Eclipse.org. For those used to working with the vagarities open source software, the MyEclipseIDE download is a real pleasure to install and use, as it installs and configures its features automatically. This feature is only the beginning of what MyEclipse has going for it. The first thing you notice is that MyEclipse adds to Eclipse *rather* than changes it. Anyone familiar with Eclipse will have no difficulty picking up and immediately using MyEclipse.

MyEclipse includes open source solutions for visual Web design, UML modeling, JSF and Struts for Model-View-Controller (MVC) development, AJAX, and object-relational mapping. Most recently, it has integrated the Matisse Swing client UI designer from NetBeans, as well as template-based Web development. Perhaps the best advantage of MyEclipse is the ability to take the integrated platform and continue to customize it to meet even more specific needs. (For more information about the NetBeans IDE, see the sidebar, "What About NetBeans?")

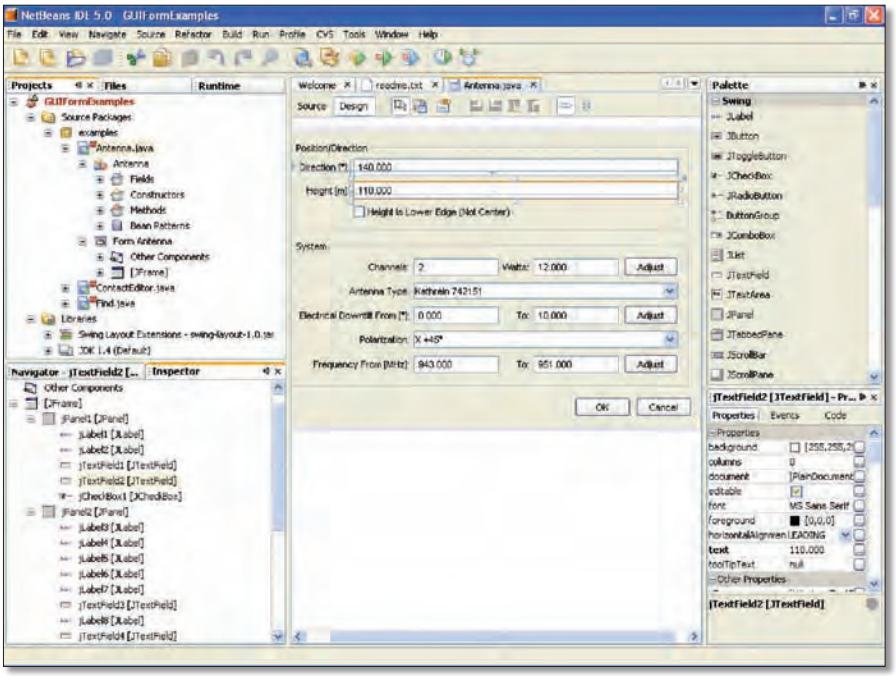


**Figure 4 | Forms Design** The NetBeans Matisse forms designer provides a realistic view of the form and the tools needed to get the design right the first time.

Eclipse itself isn't a part of the MyEclipse install, so you have to download and install the Eclipse platform prior to adding MyEclipse. MyEclipse incorporates an automated installation routine that takes care of updating the menus and adding options in the Preferences for the new components, and it simply works. (I've attempted to add a variety of plug-ins to Eclipse on my own in the past; the mechanics certainly couldn't be easier, but in reality there are often conflicting versions of prerequisite plug-ins that can make it tricky or just impossible to get certain components to work together.) Within the scope of its added functionality, MyEclipse enhancements cause no problems to the Eclipse platform as a whole.

While working with open source has often been difficult because of limited user interface facilities, MyEclipse and of course Eclipse itself make it easy to use in the development process. The sample application was built within the same amount of time it took with the best of the other products. The combination of open source tools, added features, and support for emerging standards make MyEclipse a productive alternative, no matter what the underlying platform.

Price is a consideration in looking at productivity, and Genuitec prices MyEclipse on a subscription model at $30 per year. What that buys you, in addition to the elegant installation routine, is support and updates during that period, along with the integration testing and documentation that is often lacking in open source distributions. You also get the integration that makes these open source components work together seamlessly.

You could try to put a package such as MyEclipse together yourself, but you almost certainly wouldn't succeed. First, it would be difficult to match the enhancements to installation and documentation provided by MyEclipse, and monitoring various open source sites for updates and patches can be a time-consuming exercise in and of itself. Doing it yourself might cost a few dollars less in product, but you wouldn't get nearly as much out of the end result.

## Productivity Is Key

Firm conclusions are always difficult to arrive at in product testing and comparison. Strict feature comparisons put emphasis on quantity rather than quality, especially when the total number of features may hinder, rather than help, productivity. Likewise, productivity can be lost when specific features are added because those features may only be useful to a small number of developers.

Nevertheless, it is still possible to make some generalizations surrounding productivity. Productivity encompasses low costs (both purchase price and cost of ownership), along with a feature set that is useful for the majority of developer tasks and accelerates the accomplishment of those tasks. Both JBuilder (in the most useful Enterprise Edition) and Rational Software Architect can carry a significant price tag—in the thousands of dollars per developer. While they use good modeling and quality tools to accelerate the building and testing of applications, the cost of doing so reduces their overall value. Both are popular products with known value, but the cost is difficult to justify today.

At the other end of the spectrum, MyEclipse, WebLogic Workshop, and JDeveloper are freely available, or nominally priced. If you're working with an Oracle database, JDeveloper clearly delivers the highest level of productivity, and its freely available nature clearly demonstrates substantial value on this platform. The JDeveloper features that tie the IDE explicitly to the Oracle database provide shortcuts and proprietary enhancements that make several complex activities simple and fast. Beyond Oracle, however, JDeveloper lacks the flexibility and openness to add significant value to most development efforts.

MyEclipse has a unique model that requires careful consideration in developer productivity. It might be argued that any cost to MyEclipse reduces value because its components are primarily open source. However, that is a simplistic view; Genuitec adapts the open source code that it uses to work well together, wraps it so that installation is seamless, and provides capabilities not available in the original open source code. The cost of any individual development team to perform even a part of this work would be prohibitive. By spreading that cost among thousands of developers, it becomes economically feasible to provide significant additional value. JP

**Peter Varhol** is a senior member of the technical staff for Progress Software, and has a consultancy relationship with Genuitec. Contact Peter at **peterv@mv.mv.com**.

# Sybase WorkSpace: Do Something More Interesting

### A SINGLE ENVIRONMENT OFFERS WHAT YOU NEED

Sybase WorkSpace is a Java™ toolkit offering the five most important design and development tools in an integrated easy-to-use, open source framework:

- database development
- web application development
- services-oriented development
- mobile development
- enterprise modeling

### MAKES DEVELOPMENT AND DESIGN EASY TO LEARN AND USE

Using industry-leading integrated model-driven design, visual development and task-based wizards, Sybase WorkSpace automates mundane tasks and cuts the typical development tool learning curve, freeing developers to concentrate on what's important—business logic.

### FREEDOM TO CUSTOMIZE BASED ON YOUR BUSINESS NEEDS

WorkSpace's modular packaging allows enterprises the flexibility to decide how to assign business critical tasks to developers, rather than requiring them to fit into ill-fitting "roles" predetermined by a vendor. You buy only the pieces you need, enabling you to customize your environment as you see fit.

With Sybase WorkSpace, you're finally free to do the design and development that's interesting to you. For more information and to download White Papers and an evaluation copy, visit **www.sybase.com/workspace**



SYBASE®

# Cleaning a Complex Java Code Base

## Standard checks and unit tests for every line of code might be impractical, but here's a strategy for delivery expediency

by Matt **LOVE**

Checking coding standards and unit testing would be performed ideally on every piece of code before it was added to a team's code base. However, doing so is not always practical. Many organizations do not provide developers the time and resources required for testing at this level. Moreover, most organizations do not develop applications *from scratch* by writing new code for all required functionality. Rather, they typically make incremental enhancements to a large amount of functioning legacy code or add their own code to extend third-party or open source packages. The resulting code bases could include legacy code written within the organization, code obtained through a merger or acquisition, code obtained from an outsourcer, or code that was developed by the open source community and downloaded from the Internet.

Consequently, most teams accumulate large and complex code bases with at least some code that has not been subject to coding standard analysis and unit testing. This accumulation involves several critical risks. When the application is used in a way that development and QA didn't anticipate (and didn't test), the code might throw unexpected run-time exceptions that cause the application to become unstable, produce unexpected results, or even crash. The code also might open the only door that an attacker needs to manipulate the system and/or access privileged information. Small coding mistakes could lead to significant performance or functionality problems. The code's functionality might be broken as the application evolves over the course of its life cycle.

If your team already has a large and complex code base (hundreds of thousands, or even millions, of lines), it's not too late to benefit from coding standard analysis and unit testing. As long as these practices are automated and applied properly, they can still be used to identify functionality, reliability, security, and performance problems before release and deployment—as well as to satisfy any contractual obligations for performing unit testing or complying with a designated set of standards.

Let's look at a simple two-step strategy that has been proven to deliver fast and significant improvements to large and complex Java code bases. The first step is using coding standard analysis to identify bugs and bug-prone code. The second is using unit-level regression testing to ensure that the functionality is intact and using unit-level reliability testing to ensure that all code base changes are reliable and secure. Both steps can be automated to promote a consistent implementation and allow your team to reap the potential benefits without disrupting your development efforts or adding overhead to your already hectic schedule.

### Bugs and Bug-Prone Code

Why is it important to identify bugs and bug-prone code? Complying with coding standard rules is a proven way to achieve key benefits that we can put into four groups: 1) detect bugs or potential bugs that impact reliability, security, and performance; 2) enforce organizational design guidelines and specifications (application-specific, use-specific, or platform-specific) and error-prevention guidelines abstracted from known specific bugs; 3) improve code maintainability by improving class design and code organization; and 4) enhance code readability by applying common formatting, naming, and other stylistic conventions. Rules that provide the first benefit will be referred to as group 1 rules; rules that provide the

### 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**

**JP0602** Download all the code for this issue.

**Read More**

**JP0602ML_T** Read this article online.

**JP0501NC_T** Read the related article "Why Coding Standards?" by Nigel Cheshire.

**JP040818AK_T** Read the related article "Verify Java App Development on Linux" by Adam Kolawa and Jeehong Min.

**SR_TEST_CP** Read the related article "Tips for Integrating Optimization into the Development Cycle" by Chris Preimesberger.

**Listing 1** Follow the Rule

```
public class LoginServlet extends javax.servlet.
  http.HttpServlet {
  public void doPost(
    javax.servlet.http.HttpServletRequest request,
    javax.servlet.http.HttpServletResponse response)
    throws javax.servlet.ServletException,
    java.io.IOException {
    if (validate(request.getParameter("name"),
      request.getParameter("password"))) {
      StringBuffer message = new StringBuffer();
      // rule violation
      message.append("Welcome "); // 8 chars
      message.append(request.getParameter("name"));
      // up to 20 chars
      message.append(" to the ACME online bank.");
      // 25 chars
      request.setAttribute(
        "welcome_message", message.toString());
      this.doForward(
        request, response, "/mbWelcome.jsp" );
    } else {
      request.setAttribute(
        "login_error", "invalid login name");
      this.doForward(
        request, response, "/mbLogin.jsp" );
    }
  }// doPost

  private boolean validate (
    String name, String password) {
    if (name.length () > 20)
      return false;
    return checkPassword (name, password);
  }// validate
}
```

Note the "specify an initial StringBuffer capacity" rule. StringBuffer allocates only a 16-character buffer by default, and if that capacity is exceeded, the StringBuffer class allocates a longer array and copies the contents to the new array.

second benefit will be referred to as group 2 rules, and so on.

As an example of why it's important to check coding standards even after the code is already written, assume that the analysis revealed that code for a Web application's servlet violates the "Specify an initial StringBuffer capacity" rule (see Listing 1). StringBuffer allocates only a 16-character buffer by default; if that capacity is exceeded, the StringBuffer class allocates a longer array and copies the contents to the new array. By identifying and correcting this violation, all those allocations, copies, and garbage collections are avoided, and the code is optimized. Because this servlet is used repeatedly in the application, this optimization will have a significant effect on overall application performance.

Symptoms of this problem probably could have been uncovered if the team performed an extensive amount of profiling or load testing, but tracking it to the responsible line of code would have required even more time and effort. Using an automated code analysis tool, the problem's exact source can be detected automatically in seconds, without writing team members to write a single test or manually track down the root cause of the slow performance.

To determine what's required decide which coding standard rules to check. First, review industry-standard Java coding standard rules, and decide which ones are most applicable to your project and will prevent the most common or serious defects. For instance, if your project is using technologies such as JDBC, Enterprise JavaBeans (EJB), or JavaServer Pages (JSP), review and select rules designed specifically for these technologies and general Java coding standard rules. The rules implemented by automated Java code analysis tools offer a convenient place to start for general rules that can improve reliability, security, and performance. For example, some rules many teams choose to enforce include:

- *Reliability rules*: Avoid dangling else statements; avoid try, catch, and finally blocks with empty bodies; and do not assign loop control variables in the body of a for loop.
- *Security rules*: Do not compare Class objects by name, do not pass byte arrays to DataOutputStream in the writeObject() method, and make your clone() method final for security.
- *Performance rules*: Close input and output resources in finally blocks, prevent potential memory leaks in ObjectOutputStreams by calling reset(), use String instead of StringBuffer for constant strings, and use StringBuffer.append() instead of + to concatenate strings.

Also, consider rules that are unique to your organization, team, and project (for instance, an informal list of lessons learned from past experiences). If needed, you can supplement these rules with the coding standard rules listed in books and articles by Java experts.

Consider these questions as well: Do your most experienced team developers have an informal list of lessons learned from past experiences? Have you encountered a specific bug that can be abstracted into a rule so that the bug never occurs in your code stream again? Are there explicit rules for formatting or naming conventions that your team is required to comply with?

## Looking Back

Because legacy code bases are typically very large, checking a legacy code base requires a special strategy. It's important to recognize that legacy code's design and development rule compliance will not be consistent because different parts of the code base probably originated from different sources. Applying rules from groups 3 and 4 to the entire code base is likely to result in an impractically large number of rule violations that might be more overwhelming than helpful at this stage of the project. An initial focus on rules from groups 1 and 2 for legacy code checking is strongly recommended. This focus will identify significant problems that should be corrected before the release and deployment.

Let's look at automatically checking the code base and responding to findings. Manually checking whether a large and complex code base follows coding standard rules would be incredibly slow, resource intensive, and error prone. Even if you had the vast resources required to manually review the code base, some rule violations would be overlooked inevitably, and just one overlooked rule violation could cause serious problems.

A more practical, thorough, and accurate way to check whether a large code base complies with coding standard rules is to

use an automated coding standard analysis tool to check the entire code base at a scheduled time each night. There are two complementary strategies that are well suited to the nature and size of legacy code: smoke alarm mode and gradual "fix it" mode.

In smoke alarm mode run a smaller rule set (including only groups 1 and 2 rules) on the entire code base to check if the code has critical problems. If violations are found, treat them as bugs (fix them immediately). In gradual "fix it" mode select a code module, run a full rule set on it, and then fix and/or refactor the code as needed. This mode is used to improve general compliance. Be sure to use it to check all new and modified code. If possible, check that code is compliant immediately after it is written and before it is committed into source control.

It's also possible that different modules in the legacy code base call for different rules, especially from group 2. For instance, some code analysis tools allow users to apply a filter to enable or disable a specific rule or a group of rules for a given set of files, which allows such custom-tailoring of the rules to the nature and origin of the code. This filtering can be thought of as *file-based* or *directory-based* application of specific rules.

Let's turn to the second part of the strategy, using unit-level regression testing to ensure that the functionality is intact and using unit-level reliability testing to ensure that all code changes are reliable and secure. The next step toward reliable and secure code is to perform unit-level regression testing on all existing code, and then perform unit-level reliability testing (also known as *white-box testing* or *construction testing*) on any code that is added or modified. Regression tests capture existing functionality and don't report any errors until a code modification changes that functionality. Reliability tests use an unexpected stimulus and report any errors immediately. In Java, this test involves exercising each method as thoroughly as possible for both categories of tests and checking for uncaught run-time exceptions in reliability tests.

## Functionality Protection

The second part of this strategy is important because a large base of legacy code is a huge investment of time and resources. Its functionality needs to be protected from undesired changes if some of that code is modified. After obtaining a certain level of acceptance, it is critical to not go backward by introducing bugs in functionality during maintenance of legacy code.

However, if your testing only checks expected functionality, you can't predict what could happen when untested paths are taken by well-meaning users exercising the application in unanticipated ways—or taken by attackers trying to gain control of your application or access to privileged data. It's hardly practical to try to identify and verify every possible user path and input or analyze every possible exception from legacy code. It is important to identify the possible paths and inputs that could cause uncaught run-time exceptions in new and security-sensitive code for two reasons:

- Uncaught run-time exceptions can cause application crashes and other serious run-time problems. Uncaught run-time exceptions—exceptions that are thrown automatically by the Java run-time system when a program violates the syntax/semantics of Java—usually indicate software bugs. They typically stem from problems related to arithmetic, pointers, and indexing and can occur at any point in a program. If these exceptions surface in the field, the resulting unexpected flow transfer and potential thread termination could lead to instability, unexpected results, or crashes. Many Java development teams have had trouble with Java-based applications crashing for unknown reasons. Once these teams started identifying and correcting the uncaught run-time exceptions that they previously overlooked, their applications stopped crashing.
- Uncaught run-time exceptions can open the door to security attacks. Many developers don't realize that uncaught run-time exceptions can also create significant security vulnerabilities. For instance, a NullPointerException in login code could allow an attacker to completely bypass the login procedure.

Now let's look at what's required to do unit-level regression and reliability testing. First you have to design, implement, and execute regression test cases for the entire code base. Create an automated regression test suite that verifies whether each unit continues to function as expected when the code base grows and evolves. With complex software, even a seemingly innocuous change in one part of the application can impact other functionality.

Create a functional snapshot; run a suite of unit tests that capture the methods' current behavior, which is assumed to be correct. Ideally, the test suite will capture how the units behave as the application is exercised in realistic ways (for instance, when the use cases are executed). This test suite is essentially an executable specification. By creating this test suite, you establish a baseline against which you can compare code and identify changes.

It is impractical to manually develop the required number, scope, and variety of unit test cases to execute each branch of code when you test each class as it's completed, and it's impossible when you need to find the exceptions lurking in a large existing code base. Achieving the scope of coverage required for an effective test suite mandates that a significant number of paths are executed. For example, in a typical 10,000-line program, there are approximately 100 million possible paths; manually generating input that would exercise all of those paths is infeasible and practically impossible.

When trying to create a baseline of regression tests for a large code base, a tool that automatically generates test code is essential. Team resources can then be focused on reviewing and addressing the reported test case failures and exceptions.

## Nightly Testing

Next you must review and respond to regression test findings, and test new code for reliability. Configure the automated testing tool to unobtrusively execute the complete regression test suite—all of the baseline unit tests—each night. Each test case failure (a test case that doesn't produce the baseline outcome expected for a set of baseline input[s]) indicates a change in the code's behavior. This change may be intentional or unintentional. When code functionality changes intentionally—as a result of a feature request, specification change, and so forth—test cases related to that behavior are expected to fail because the new expected outcomes will be different than those recorded in the baseline. However, very often, other test cases will also fail unexpectedly. If so, this fail-

ure reveals a complex functional problem caused by the code modifications. If no unexpected failures are identified, you know that the modifications didn't break the existing functionality.

The appropriate response to a test case failure depends on whether the change was expected. If the new outcome is now the correct outcome, the expected test case outcome is updated, and it becomes a part of the baseline. If not, the code is corrected.

After you rerun the test, review all uncaught run-time exceptions exposed by the tests, and then address them before proceeding. Each method should be able to handle any valid input without throwing an undocumented uncaught run-time exception. If code should not throw an uncaught run-time exception for a given input, the code should be corrected. If the exception is expected or if the test inputs are not expected or permissible, document those requirements in the code, and indicate in the tool that they are expected. This procedure prevents most unit testing tools from reporting these problems again in future test runs. Moreover, when other developers extending or reusing the code see documentation that explains that the exception is expected behavior, they will be less likely to make mistakes that introduce bugs.

Now let's look at what is necessary to make your application even better without breaking it. Suppose you are safeguarding against introducing critical problems and have a complete regression suite for the software to maintain the state it needs to be for the impending release and deployment milestone. What now?

If resources permit, you have a good opportunity to continue improving the code quality. Extend your unit test suite to improve coverage, make tests more realistic, and verify the functionality specified in the requirements. Also, phase in more coding standards to identify and prevent additional coding problems. For instance, start implementing rules that improve code maintainability by improving class design and code organization, and rules that enhance code readability by applying common formatting, naming, and other stylistic conventions.

Review the coverage after running the entire test suite. If any classes received less than 75 percent coverage, customize the automated test case generation settings (for instance, by modifying automatically generated stubs, adding realistic objects or stubs, or modifying test-generation settings) so that the automated test case generation can cover a larger portion of that class during the next test run.

Identify critical modules of code that should undergo more thorough rule compliance and reliability testing. Utility code that is used from many parts of the application is the most sensitive to performance problems and unexpected inputs because that code is invoked so often in so many ways. Front-end code for user interfaces, resource loading, or other communication is the most vulnerable to security attacks, and it is an entry point into the system for unexpected input. The highest priority is establishing the baseline for protection of legacy code and putting in place a system to safeguard against new defects entering the code base. Incremental improvements on existing code should not be done until after the baseline and safeguards are in place. JP

**Matt Love** is a software development manager with Parasoft Corporation. He's been involved in the development of Jtest, Parasoft's automated code analysis and unit testing tool.

# Get Creative on the Java ME Platform

**Java ME continues to mature. Assess the platform, the standard APIs, the CLDC/MIDP stack, and device support for your needs**

by Michael **YUAN**

When Sun Microsystems introduced Java 2 Platform, Micro Edition (J2ME, which was renamed recently to Java ME) to the world in 2000, the promise was to bring Java's "write once, run anywhere" capability to the highly fragmented handheld-device market. Java ME is supposed to be the "one platform that rules all mobile phone manufacturers and carriers." It allows developers to focus their energy on creative work instead of tedious application porting across multiple devices, and it aims to create a mobile application marketplace where all applications compete on a level playing field.

After six years, Java ME has met with great success. It is now supported by all major mobile phone vendors and carriers. Today, more than one billion devices support Java ME out of the box. However, has Java ME fulfilled its "write once, run anywhere" promise?

Let's examine the current state of Java ME and the entire mobile application market. The aim here is to help you decide whether Java ME will fit your next project, and if it does, to focus on how to develop portable Java ME applications. Primarily we'll concentrate on mobile phone development on the Java ME platform, that is, the Common Limited Device Configuration (CLDC)/Mobile Information Device Profile (MIDP) stack including smartphones and PDA phones. Java ME has another stack, known as the Connected Device Profile (CDC) and Personal Profile (PP), to support larger personal digital assistants (PDAs) and set-top box devices. The CDC/PP stack has not been widely adopted and is not within the scope of this discussion.

The Java programming language is designed for cross-device portability. Java source code are compiled to a bytecode format that can be executed by the Java Virtual Machine (JVM). The JVM translates the bytecode to the native machine code for the target device at runtime. To run Java applications, a Java ME compatible mobile phone must have the JVM preinstalled. Device manufacturers develop and preinstall JVMs for their devices, and, hence, insulate the application developer from the underlying device hardware and operation system, which are typically proprietary. In fact, many mobile phones on the market have completely closed operating systems, and Java ME is the only programming interface for those devices (see the sidebar, "More on the JVM").

## Crucial Interfaces

The Java language is only the basis of Java applications. Java ME is an application development platform built on top of the Java language. The most important components of the platform are the Java ME application programming interface (API) libraries. The APIs determine what kind of applications you can develop with Java ME. To enable the cross-device portability of applications, it is crucial to standardize those APIs.

In Java ME all standard APIs are developed from the ground up as an industry consensus through the Java Community Process (JCP). The JCP membership is open to all interested vendors and individuals. Almost all mobile phone manufacturers and carriers participate in the JCP. Every API is proposed by a JCP member as a Java specification request (JSR) and then developed by an expert group. The membership of the expert group is also open. At the time of this writing, there are 68 JSRs for the Java

### 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**

**JP0602** Download all the code for this issue.

**Read More**

**JW041706MY_T** Read this article online.

**JW041706PV_T** Read the related article "Migrate Mobile Client Applications" by Phong Vu.

**JP0212RG_T** Read the related article "IDEs for Wireless Java" by Rick Grehan.

**JP0105JW_T** Read the related article "Big Plans for J2ME" by Jim White.

ME platform. Some of the most important JSRs are:

- **JSR 30** for version 1.0 and **JSR 139** for version 1.1 – The CLDC specifies the core language APIs for Java ME. For instance, it defines classes such as String and List, and it also specifies how the bytecode should be loaded into the JVM.
- **JSR 37** for version 1.0, **JSR 118** for version 2.0, and **JSR 271** for version 3.0 – The MIDP specifies basic application-level APIs. It contains a UI widget library for small screens, a set of low-level APIs to draw directly on the screen and capture-user input, a network API to send and receive data over the HTTP protocol, and a persistence API to store application data on the device memory. The MIDP also specifies the application life cycle (that is, the MIDlet model for starting, pausing, and exiting the application); security model (that is, how to determine whether an application is *trusted* to access the network, and so on); and how the application should be deployed over the wireless network.
- **JSR 120** for version 1.0 and **JSR 205** for version 2.0 – The Wireless Messaging API (WMA) provides access to the device's Simple Message Service (SMS) messaging functionalities. Using the WMA, the application can send an SMS message to any other device with a phone number. It can also receive incoming SMS messages. However, the WMA doesn't have access to the phone's native SMS inbox, and therefore cannot receive regular phone-to-phone SMS messages. It can only receive messages addressed to a special SMS port on the device. Most computer-based SMS tools and Internet-based SMS gateways allow you to send such messages with port numbers. In WMA version 2.0, you can also send and receive Multimedia Messaging Service (MMS) messages from the application.
- **JSR 135** for version 1.0 – The Mobile Media API (MMAPI) provides access to the device's audio and video peripherals. You can use the API to play back audio or video clips. On some devices, you can also use the API to record voice and capture picture/video from the on-device camera.

## To the Higher End

All Java ME-compatible mobile phones support at least CLDC and MIDP. Most devices on the market today support WMA and MMAPI as well. A mobile phone with CLDC, MIDP, WMA, and MMAPI support can be labeled as a Java Technology for Wireless Industry (JTWI)-compatible device. In addition, many high-end Java devices also support one or several of these optional APIs:

- **JSR 75** – The personal information manager (PIM) and file connection optional package provides access to the device's native PIM databases—for example, todo list, calendar items, and address book. This API also allows the application to save files to the device's native file system, as opposed to the simulated persistence store defined in MIDP.
- **JSR 172** – The Web services API provides a lightweight XML and SOAP parser library. You can develop mobile clients for SOAP Web services using this API.
- **JSR 184** – The Mobile 3D API is a lightweight 3D graphics library. It allows the application to create a virtual world and manipulate objects in that world. It is an important API for mobile game developers.
- **JSR 179** – The Location API allows the application to figure out the device's current location through an on-device, GPS receiver or through a query to the carrier's location server. When combined with mapping data—for example, Google Maps and Yahoo Maps—the location API allows us to develop powerful, location-based applications.
- **JSR 82** – The Bluetooth API provides access to the Bluetooth radio on the device. You can use the Bluetooth API to exchange data objects and/or simulate serial data links between nearby devices.
- **JSR 180** – The Session Initiation Protocol (SIP) API supports the SIP for network applications. SIP is important for push-based applications. It could also potentially open the possibility for Voice over Internet Protocol (VoIP) clients on mobile phones.
- **JSR 177** – The Security and Trust API provides access to the mobile phone's SIM card. The SIM card uniquely identifies the mobile subscriber account on the network. Through this API you can gain access to data and applications stored on the SIM card.

The optional APIs allow Java ME to scale from very low-end devices to high-end smartphones without falling into the trap of the *lowest common denominator*. Basic MIDP applications run on all Java ME devices. Applications designed for high-end devices can take advantage of the more capable hardware (for example, camera, GPS, and Bluetooth radio) through those optional APIs. Those applications probably wouldn't run on low-end devices (nor should you expect them to), but the point is that they are portable across similarly equipped devices from different vendors. With so many available APIs, Java ME is a comprehensive platform for developing all types of mobile phone applications, and yet it preserves the cross-device application portability as we can reasonably expect.

## API Support by Device

For the APIs to be useful, the device manufacturers must implement and support them on the devices. One of the great successes of Java ME is its wide adoption among device vendors. Now, let's check out what Java ME APIs are supported on several popular mobile phones.

Sony Ericsson has a great line of Java phones. They have large heap memory space, large flash storage space, and no limits on the

### More on the JVM

The JVM is much more than just a cross-platform layer for interpreting Java bytecode. It provides automatic memory management, run-time optimization, and a security sandbox for applications. When a Java application crashes, it crashes inside the JVM and won't affect other applications on the same device. A Java application cannot access any device resource without the JVM permission. All of these aspects are crucial productivity features that make Java a popular programming language. The stability and security features provided by the JVM are especially important for mobile phone applications.

size of the Java application (JAR file). On a Sony Ericsson phone, the Java ME application can make outward HTTP connections through the WAP channel. Therefore, you need only a cheap WAP data plan (for example, the $5/month unlimited t-zones plan from T-mobile) to use networked Java ME applications. That is a huge plus for many because phones from other vendors often require purchasing the full "Internet data plan" ($20/month for T-mobile) to use the network (TCP/IP) in Java applications.

A midrange smartphone device like the K700 supports these Java ME APIs: CLDC 1.1, MIDP 2.0, Mobile 3D API, WMA, and MMAPI. The support for Java 3D on midrange devices (and even mass-market devices like the K300) is great for game developers too. However, it's also noteworthy that this device doesn't support the PIM and file connection API. It also lacks support for the Bluetooth API, although it does have Bluetooth radio.

A high-end Sony Ericsson phone like the W900 walkman phone is a truly powerful Java ME device. It supports all the APIs supported in the K700, plus Bluetooth API, PIM and file connection API, and Web services API. The Mobile 3D API support in W900 is backed by hardware acceleration. You can develop very nice and fast 3D applications for the W900.

Nokia is the biggest mobile phone manufacturer in the world. It also sets the standard for mobile phone features and UIs. If you are developing a Java ME application, you will probably target a Nokia phone for prototype development at first. Nokia is a key member in the JCP, and it drives the development of many Java ME APIs.

A low to midrange Nokia Series 40 phone like the Nokia 6230 typically supports these APIs: CLDC 1.1, MIDP 2.0, WMA, MMAPI, and the Bluetooth API. A popular Nokia S60 smartphone like a Nokia 6680 supports CLDC 1.1, MIDP 2.0, WMA, MMAPI, Mobile 3D API, Bluetooth API, and the PIM and file connection API. A high-end Nokia S60 device like the E70 supports all of the aforementioned APIs plus the Web services API, Security and Trust API, Location API, and the SIP API.

Like Nokia, Motorola is an early supporter of Java ME and is a key JCP member. However, Motorola phones' Java support has left a lot to be desired. For instance, Motorola's best selling RAZR V3 phone supports only CLDC 1.1, MIDP 2.0, WMA, MMAPI, and proprietary APIs to access the address book, file system, and the fancy LED lights on the phone. It's a pity that such a slick and popular phone doesn't support some of the more advanced and standard APIs.

## Pros and Cons

Research In Motion's BlackBerry is a wildly popular e-mail device among enterprise users. The entire suite of software on the BlackBerry handset is built using Java. BlackBerry supports CLDC 1.1, MIDP 2.0, and an array of proprietary APIs to access the device's native e-mail client and other PIM databases. Those APIs are highly useful in constructing enterprise applications over BlackBerry's push messaging platform.

Palm Tungsten/Treo and Windows Mobile devices typically do not come with the JVM preinstalled. You can download and install third-party JVMs yourself to run Java applications on those devices. However, the third-party JVMs are typically limited to CLDC 1.1 and MIDP 2.0 support without any optional API package support.

While Java ME is a highly successful platform for mobile applications, several shortcomings that hinder its adoption over the past six years have been observed. For developers it is very important to understand those shortcomings and how they might affect your development projects. Let's take a look at some potential solutions to those problems.

Device *fragmentation* refers to the reality that different devices support a different set of Java ME APIs and have different behaviors even under the same API. Fragmentation breaks application portability, and it is one of the biggest complaints from Java ME developers. However, it is important to understand that there is nothing wrong with *fragmentation* per se. Mobile devices are personal and specialized devices; different customers require different devices. It's a good thing that device manufacturers make a variety of devices to address the diverse market needs and differentiate themselves from competitors. In fact, this type of fragmentation is a sign of innovation. What's missing is a universal and standard best practice to help you work with fragmentation. Typical sources of fragmentation include different devices having different:

- Hardware add-ons and, hence, support different Java ME optional APIs – For instance, a low-end device without Bluetooth radio would not support the Bluetooth API.
- Priorities for their storage place and other computing resources – For instance, a device for the youth market probably supports the Mobile 3D API, but it is unlikely to support the Web services API.
- Form factors and different screen resolutions – They also support different data input methods (that is, keyboard, keypad, touch screen, voice recognition).
- Application sizes – Some devices can only install applications smaller than 100 KB, while others permit up to several MBs. They also support different amounts of heap memory space, persistent storage space, and maximum number of concurrent threads.
- Implementations of the same API – For instance, two devices might both implement the MMAPI, but one device supports capturing video clips and MP3 playback while the other only supports simple MIDI playback.
- JVM implementation bugs or behavior when the specification is vague – A major source of confusion comes from the multithread behavior of different devices. This behavior could be an issue when you have several threads updating the screen for animation and retrieving data from the back-end server. Different devices also have different behaviors when you try to free memory space by running a garbage collection.

## Provider Assistance

Mobile application developers typically develop one application that runs well on a popular device, and then try to port the application to other devices in the same class. Device manufacturers can help by grouping similar devices together. For instance, Nokia groups all of its devices into three developer platforms (Series 40, S60, and Series 80). Devices on the same

# THE ARCHITECTURE JOURNAL™

Input for Better Outcomes

Come visit the Journal's new home at **www.ArchitectureJournal.net**. The new site contains a full library of articles from previous Journal issues in addition to upcoming highlights of our next issue. Browse the content today and post comments and letters directly to the editor!


**Now live at www.ArchitectureJournal.net!**

Microsoft®

ARC

platform have similar screen sizes, hardware capabilities, and support similar Java ME APIs. You need to develop your application for a representative device in each platform, and then only minor changes are required to port them to every device on the platform.

In the porting process, you typically need to optimize resource files (for example, images and sound clips) for the target device's screen, speaker, and memory space; add or remove functionalities based on the API availability on the target device; and provide source code-level workarounds for JVM bugs or other low-level JVM differences.

Several third-party solutions have been developed to address the device fragmentation problem. For instance, the NetBeans Mobility Pack (see Resources), which is a premier, free IDE for Java ME, supports precompile conditions embedded in Java code as comments. You can choose to include and/or exclude certain code blocks for each build target. It is a very powerful way to introduce minor code changes between target devices.

By tweaking the build script, you can also choose what resource files to include for each build target. If you do not want to deal with the application porting issues by hand, Tira Wireless develops an automatic tool for porting and optimizing Java ME applications. The Tira Wireless Jump suite has a very comprehensive database that documents differences among devices (see Resources). You can simply feed your "reference implementation" for a popular device into the Jump suite, and it will make changes automatically to the code and resource files to generate applications for the target device.

The second major shortcoming of Java ME is that it's originally designed without much thought about *mobility*. In fact, the CLDC/MIDP stack looks very much like a miniature desktop environment with UI widgets tweaked to fit the small screen. Furthermore, because of the Java security model Java ME applications do not have any access to device functionalities not exposed as Java APIs. As a result, most Java applications are limited in the CLDC/MIDP sandbox, and they are distinctly different from native applications on the device because of the lack of integration with the underlying system.

There is no integration with the device's native applications (for example, the messaging client, the video recorder, the music player, and the screen saver), and there is little integration with low-level hardware features (for example, access to the device serial number, cell ID, and so on). The CLDC/MIDP sandbox is probably okay for simple, form-based business applications or simple games ported from the PC world. However, the problem is that there is only limited need for "desktop replacement" mobile applications. Many users already use laptop PCs or tablet PCs for this type of application. Plus, those Java ME applications only represent incremental improvements over WAP browser and Flash-based applications. They aren't all that exciting, and therefore the adoption rate is low.

## Wish List

What mobile application users and developers really want are applications that can truly take advantage of mobility features that are available only on mobile phones. We want applications that integrate tightly with the underlying phone platform and behave like native applications. For instance, here are several feature examples that would be great to have in Java ME applications or games: the ability to make, receive, and manage voice calls; the ability to make use of and manage users' personal data on the phone (for example, address book, calendar, photos, and ringtones); an idle screen or screen saver to run in the background while processing user input and SMS or Bluetooth for responding to incoming messages; the ability to uniquely identify the user through IMEI number, subscriber number, or even digital certificate; location sensitivity; and camera-based applications.

Of course, the Java ME optional API packages are designed to provide Java applications more access to the underlying platform. For instance, the PIM API allows access to the PIM database (for example, address book) maintained by native applications, the file connection API provides access to the photo and music folders on the device, and the Location API enables location-based applications. However, some important features are simply not supported in current APIs. For instance, there is no Java ME API to support voice calls, which is by far the biggest application for mobile phones. Even for features that are supported in current APIs, the JVM implementation often leaves a lot to be desired. For instance, the photos captured from the MMAPI typically have much lower quality than photos captured from the native camera application.

To make Java ME a better mobile application development platform, we need to push out more optional API packages and get them implemented by phone manufacturers, which leads to the next weakness in Java ME.

As mentioned previously, all of the Java ME APIs are collaboratively designed by the JCP. Many JCP members compete among one another. The JCP process certainly helps those vendors to reach a consensus that they can all support. However, this *design-by-committee* approach is also very slow, especially when some vendors have political agendas. For instance, it took the JCP more than three years to develop the PIM and File Connection optional packages in Java ME, which is a very long time in the world of mobile applications. That delay has resulted in the situation today in which the Java environment on the majority of Java ME devices has no integration with the most popular native applications.

A potential solution for this problem is to encourage mobile phone vendors to develop and support proprietary Java APIs on their devices if no standard APIs for the same functionalities are available. Nokia and Motorola have used this proprietary API approach in the early days of Java ME out of necessity. BlackBerry is still doing using this approach today with very good results. Applications developed against those proprietary APIs would not be portable. However, that would give Java developers a way to write advanced applications for this particular device (or family of devices) if they choose to. In fact, the proprietary API can also act as a testing ground for JCP APIs. If those APIs are proven successful, the vendors can then work together to make it a standard. JP

**Michael Yuan, Ph.D.**, is a developer, speaker, and author specializing in end-to-end enterprise and mobile solutions. Michael is the author of three mobile technology books including *Nokia Smartphone Hacks* (O'Reilly Media Inc., 2005); *Enterprise J2ME* (Prentice Hall PTR, 2003); and *Developing Scalable Applications for Nokia Series 40 Devices* (Addison-Wesley Professional, 2004). He has served as an expert group member in several Java Specification Requests (JSRs), and he works currently for JBoss Inc.

# Java's Desktop Comeback

by Peter **VARHOL**

## New vertical market applications require customization, and the RCP may provide the best tools for the job

Eclipse has one. NetBeans has one. Eclipse claims to have made significant strides in getting developers excited about the technology and using it in development efforts. I'm not referring to a freely available open source IDE, of course, but rather the Rich Client Platform (RCP).

You can think of the RCP as an application framework. How does it work, you might ask? Both Eclipse and NetBeans are fundamentally IDEs, looking and behaving the way developers expect them to perform. While Eclipse has shifted its image over the last couple of years to that of a more generic application development platform, the concept of the RCP doesn't even seem to fit under that umbrella.

Further, the recent promotion of the Java rich client seems odd, coming as it does after the development community seems to have determined that Java is best suited for Web applications and middleware. Once upon a time, in the dawn of the Internet era, Java was in fact seen as primarily a platform for visual experiences. This notion was supported by a rich set of layout managers that in theory enabled developers to deploy the same UI on different display types such as desktop computers and cell phones.

However, Java on the desktop or in applet form suffered from poor user interface (UI) controls, inconsistent layout managers, and above all, poor performance. Within two years, Java largely disappeared from the desktop, and still later Java 2 provided the features needed for true enterprise back-end solutions. Except for Web applications, Java almost disappeared from the desktop.

### Ch-Ch-Changes

What has changed? Fortunately, plenty. Here are probably the three most important changes we've experienced in the platform in the last ten years.

1. *Managed languages are mature*. Let's face it, Java was painfully slow in the 1.0 time frame. Bytecode was fully interpreted, whereas today just about everyone JITs it. Likewise, early versions of the Java Virtual Machine (JVM) were not tuned for performance. The concept of managed languages was new for most, and expectations were based on the relative performance of C and C++ applications. Java clearly suffered as a result. However, today managed languages are the mainstream. Microsoft has introduced its own managed platform, and it is no longer such a radical idea. Scripting languages such as Python and Ruby, which are interpreted, are accepted as solutions that provide for sufficient performance and scalability on many different applications. Running Java on the desktop is no more foolish than running Microsoft on the desktop.

2. *Computers and networks are faster*. The poor performance of Java was exacerbated by slow desktop computers. While they may have seemed pretty fast at the time, the average computer in 1996 ran at about 133 MHz and had about 8 MB of memory. That amount was insufficient memory to contain an adequate working set for both a JVM and an application, so the result was a lot of disk swapping and waiting. Networks had similar limitations, especially with Internet access. Anyone who tried running Web applets across dial-up networks around 1996 knows that download speeds were painfully slow. In contrast, today the proliferation of T1 lines, frame-relay networks, DSL, and cable modems make Internet access blazingly fast, and for LANs, gigabit Ethernet is fast becoming the standard.

3. *User interface choices are better*. When Java first launched, its UI controls were primitive and without many properties that developers expected to have under their control. On the Eclipse platform, the Simple Widget Toolkit (SWT) provides the Windows look and feel that is familiar to most computer users, and on NetBeans developers are more than happy with Swing.

Thus, it is not only possible, but probably inevitable, for Java to make a comeback on the desktop. While rich client applications seem to have lost some of the enthusiasm of developers and IT administrators, application end users still prefer the feel and interactive nature of the desktop client.

However, those charged with building, maintaining, and administrating rich client applications generally don't like to work on them. From the standpoint of

### 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**

**JP0602** Download all the code for this issue.

**Read More**

**JP0602OE_T** Read this article online.

**JP0601OE_T** Read the related article "Fast or Good?" by Peter Varhol.

**JP0511OE_T** Read the related article "One or the Other" by Peter Varhol.

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

developers, there is too much baggage to bring along. They not only have to create the required features and flow of work but also housekeeping activities such as window and text manipulation. As the application is maintained and enhanced over its lifetime, adding new features while maintaining the quality of existing ones becomes more and more difficult because old and new code becomes intertwined.

## Simply Plug It In

From the standpoint of system and software administrators, rich client applications are the bane of their existence. Installation requires either complex scripts or visits to every desktop, and solving problems often requires diagnosis directly on the client. Eclipse addresses the baggage and upgrade issues through its unique plug-in and update strategies. The plug-in strategy provides for features to be incorporated as separate *plug-in* modules to simply be placed in the correct directory for those features to be recognized and integrated into the platform.

This strategy accomplishes two purposes. First, it provides a framework of windows, menus, graphics, and other UI elements for the plug-in. In fact, it goes deeper than just the UI; it also provides underlying communication and resource management foundation for an application. Writing the features of the application is an easier and more straightforward proposition. Second, it provides a streamlined way to install and support that application. Many enterprises keep a set of standard images for their desktops, so that a given system configuration can more easily be created on demand. If those images also include an RCP, then installing the right applications for a specific user can consist of loading the correct plug-ins.

Eclipse also goes one step further with its Update Manager, a method by which new versions of plug-ins can be downloaded from a specified location and installed automatically. It is really a straightforward HTTP link to a given download site, but Eclipse automates the process by either looking at features already installed or letting you set the link to the application you want to install. Either way, this feature makes it possible to load up that newly imaged computer with the correct application by simply entering the correct URL into the Update Manager. Rich client application deployment and maintenance just got significantly easier.

The RCP concept may make more sense in vertical industry applications, where customization of features and workflow is an important part of the process. It is much easier to maintain multiple versions of plug-ins rather than multiple versions of entire applications. Yet, isn't it still an IDE at heart? Well, yes, but at some level an IDE is simply an application for building applications. It shares more characteristics with its end product than we might realize. Virtually all computer users do input, editing, switching between files and windows, and running tools no matter what application they are working with, and those are the capabilities that come with the platform, whether it is Eclipse or NetBeans.

> The plug-in strategy is really quite unique, and it represents an entirely new way of thinking about application development

The greatest strength of the RCP may also be its greatest weakness. One of the key advantages in getting an application accepted by end users is to have a look and feel that is familiar to those users. Being able to leverage the look and feel of the RCP across multiple applications is an incredibly powerful incentive to adopt such a standard platform.

But the weakness is that the RCP look is not the look that users are familiar with today. That distinction belongs to Microsoft Office, which is by far the most widely used rich application today. Most applications try to mimic that look, reasoning that their users will have a more positive initial impression of the application and require less training to use it effectively. In effect, to get an application accepted, often the best strategy is to look like other popular applications.

## Past Perceptions

Because Java rich client UIs are only starting to emerge, their look is unfamiliar to most users. Here the Eclipse RCP may have a slight advantage, since the SWT UI controls use the Windows look and feel, and thus may be more familiar to more computer users. Moreover, more software developers use Eclipse, so its look is becoming familiar to that important group of users. Either way, it's an uphill battle for any look and feel not associated with Windows and Office to gain popularity among end users—uphill, but not impossible.

Despite all of its apparent advantages, I confess that I still have my doubts about the viability of the RCP concept. Some of those doubts are rooted in what are probably out-of-date biases toward rich Java applications. I don't know if the RCP can change that longtime perception that Java isn't a language for rich clients.

The plug-in strategy, however, is really quite unique, and it represents an entirely new way of thinking about application development. Developers I've talked to have mixed reactions; they tend to like the platform idea in theory, but few see it as something they could realistically put to use. For the most part, they are not building entirely new applications, but rather are adding features to, or updating, existing ones. The problems with bringing an old code base forward are well known to all of them, but none of them can justify starting over with an entirely new code base.

This outlook may be the biggest impediment to RCP, but new vertical market applications are being written on a regular basis, as markets shift and regulations prompt changes. Because vertical market applications often need customization, the RCP may be more compelling in this type of development environment. I applaud the trend, and believe that over time it will mark a watershed in how we build applications. JP

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

# The Two Schools of Lazy Programming

## Apply a different metric to adopt practices that will save you time while achieving a desired result

by Daniel F. **SAVARESE**

During my career, I've had the opportunity to evaluate software development practices at all types of organizations. From small start-ups and large corporations to government and academic research labs, software projects face many of the same problems. Unfortunately, many of those problems are self-created and therefore avoidable. Strangely, self-created problems are caused often by laziness and also can be solved by laziness. The lessons taught by these two schools of lazy programming—the bad and the good—can help open the doors of productivity. But how do you tell the difference between bad laziness and good laziness? There's the rub.

Recently, I had the opportunity to evaluate software development at a very early stage start-up. The name of the game at start-ups is speed. You've got to code fast and get product out the door so you can start making money and capture a piece of the market before your competitors bury you or your money runs out. Cutting corners is common practice, but in the long run does more harm than good. Start-ups tend to succeed in spite of themselves, not because they are models of efficiency.

The start-up I evaluated fit the pattern I've encountered all too often at companies big and small. No revision control. No release management process. No requirements documents. No design documents. No API documentation. No test procedures. You get the picture. Worse yet, the CTO understood that the company was cutting corners and offered the usual defense of not having enough time to do things right. Many compromises can be forgiven if they produce good results, but eschewing good practices knowingly in the interest of saving time in the short term will always cost more time in the long term.

The first commandment of software development should be "Thou shalt not program without a revision control system." Whether it's a one-programmer project or a hundred-programmer project, revision control is the foundation for creating reproducible results in software development. Single programmers cannot recover from their mistakes and failed experiments without revision control. Efficient multideveloper collaboration is not possible without revision control. Reliable release management is not possible without revision control. Yet many companies develop software without it.

### No Excuses

Projects avoid revision control because of the bad kind of laziness. An individual programmer starts coding on his or her own, as happened at the start-up I mentioned, and becomes more concerned with programming than configuration management. In the absence of a preexisting configuration management infrastructure, it seems less time consuming to simply code away. But then the programmer alters some code and changes his or her mind, deciding it's best to revert the changes. Oops! No revision control. Now the programmer has to recreate the original version from memory and starts making back-up files every now and then before making major changes. Come release time, he or she makes another backup and gives it a release number. Suddenly, the programmer's managing an ad hoc version control system.

Truly lazy programmers would start off understanding that making manual backups of source files is tedious and error-prone. "I don't want to waste my time copying files to numbered directories," they think. "I'll just save some time and use a proper version control system." With the many free and feature-rich revision control systems available, there's simply no excuse to not take this tack. At every organization where I've had to introduce basic software development practices, the practice that has gotten the greatest positive response from programmers has been the use of revision control. Every programmer I've met who has moved from programming with no versioning system to programming with a versioning system has said the same thing: "I'm going to use version control for everything from now on. I don't know how I managed without it!" That's the good laziness setting in. If you are truly a lazy programmer, once you recog-

### 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**

**JP0602** Download all the code for this issue.

**Read More**

**JP0602PS_T** Read this article online.

**JP040818AK_T** Read the related article "3 Tips for Developing on Linux" by Adam Kolawa and Jeehong Min.

**JP0305PS_T** Read the related article "Write Tests to Refine Your Code" by Daniel F. Savarese.

**JP0310PS_T** Read the related article "JMX for Managing Java Applications" by Daniel F. Savarese.

nize that a practice will save you effort, you embrace it wholeheartedly.

Source control seems like such a basic element of software development that it is easy to disbelieve organizations exist that don't use it. Still, it must be applied effectively to provide benefits. For example, after the company in question adopted a version-controlled source code repository, it didn't have any configuration management procedures to guide project organization and release management. Therefore, interdependencies between modules and shared dependencies were handled through duplication. Approximately 100 third-party libraries were duplicated, a separate instance appearing in the repository for each dependent module. Project source files also were duplicated. For example, independent service components all shared the same configuration file format. To read the configuration file, the services required the same configuration class. Instead of placing this class in a separate library, the source file was copied into the source tree for each service. Which version was the master copy?

The rampant duplication was borne out of bad laziness. You're in the middle of coding one separately versioned component and need some functionality from another separately versioned component; therefore, you copy the code or the entire source file instead of taking the time to organize your code into reusable libraries. Now, instead of making changes in a single place, you have to apply the changes in every place you copied the code. A programmer exercising good laziness will recognize the time and maintenance savings to be derived from organizing the code up front.

Again, organizing code into units that avoid duplication seems like such a basic practice that it's easy to disbelieve any software development project would not do so.

## Testing Shortcut

Nonetheless, even if you organize your source code and build system to avoid duplication and rely on versioned snapshots of class libraries, there remain many ways to create unnecessary work for yourself. A common shortcut that should not be easy to disbelieve is a lack of test procedures. Testing encompasses more than simple unit tests, but unit tests are a good place to start. If you implement unit tests as you develop your code, you build reliability into your system as you go along. Writing unit tests as you go along is not overly time-consuming. Neither is writing API documentation.

There is a lot of dead time involved in software development, where programmers stare at the computer screen and think. It's not overly time consuming to use that dead time to both think and write API docs and unit tests. Tracking down and fixing avoidable bugs after a release is overly time consuming. Writing API documentation for hundreds or thousands of classes after you've forgotten all of the details also is overly time-consuming. Going back and writing unit tests after the fact because you finally realize you need them is overly time-consuming.

> The first commandment of software development should be, "Thou shalt not program without a revision control system."

The start-up I mentioned earlier had written absolutely no API documentation and no unit tests. Their production software was deployed as a service off of the head branch without cutting release snapshots. Their idea of testing was to deploy the product and wait for the customer complaints to arrive. The service was crashing once every couple of weeks. Without any performance measurements and analysis, they decided to make some performance enhancements. The law of unintended consequences took hold and the service proceeded to crash every few days. Different parts of the code base relied on different object persistence mechanisms, each of which created its own set of software maintenance problems.

To the company's credit, it had deployed an issue-tracking system. However, the list of issues slated for the next release was so long it could never be completed in a reasonable time frame, causing them to deploy their service off of the head branch on a regular basis. Still, they were making money as so many companies do in spite of their inefficient practices.

This company needed to cut its losses and invest the time to put its house in order instead of offering the excuse that "there's no time to do this." Small software companies often start life with one or two programmers. Single programmers should conduct software development as though they were working with other programmers. Write down requirements even if they consist of a few bullets in a text file. Sketch out your designs. Document your code as you write it. Write unit tests. Plan releases from the very start as achievable goals that focus on a handful of changes. Measure and analyze performance before you optimize code. Manage complexity by simplifying wherever possible (for example, use one object persistence system instead of four). Don't succumb to the bad laziness that is hastiness.

Any practice that appears to save you time in the short run but costs you time in the long run is an example of bad laziness. Any practice that saves you time in the long run while achieving a desired result is an example of good laziness. Use that metric to help decide what practices to adopt. You don't have to follow a particular software development process step by step. It takes time to save time. It takes work to truly be lazy. JP

**Daniel F. Savarese** is the founder of Savarese Software Research. 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, Sava Algorithms, and BareHTTP 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**.

*Java Pro*'s article index, published periodically as a convenience for readers who need specific articles, lists content that appears in print and online. Articles are organized by issue and include Locator+ codes. To read an article online, go to either the home page for Fawcette Technical Publications at www.ftponline.com, or the Java Pro Online site at www.javapro.com, and type the Locator+ code into the field in the upper-right corner of the page. This index is also available online.

| | | |
|---|---|---|
| The Philosophy of Interface-Driven Design | Jason Byassee | JP0311JB_T |
| Sun ONE Approaches to Web Development | Sameer Tyagi | JP0311ST_T |
| Object Enterprise: Write Once, Run… Where? | Peter Varhol | JP0311OE_T |
| Pro Shop: A Trio of Quadtrees | Daniel F. Savarese | JP0311PS_T |
| Editor's Note: The Ice Man Cometh | Kay Keppler | JP0311EN_T |
| Product Review: WebLogic Platform 8.1 | Daniel F. Savarese | JP0311PR_T |
| **October 2003 Vol. 7 No. 10** | | |
| *Java Pro* 2003 Enterprise Buyer's Guide | Editors of *Java Pro* | JP0310BG_T |
| The 2003 *Java Pro* Technology Roundtable | Simon Phipps | JP0310RT_T |
| Object Enterprise: The Power of Patterns | Peter Varhol | JP0310OE_T |
| Pro Shop: JMX for Managing Java Applications | Daniel F. Savarese | JP0310PS_T |
| Editor's Note: The Right Stuff | Kay Keppler | JP0310EN_T |
| **September 2003 Vol. 7 No. 9** | | |
| Architecture Is Key to Optimization | Peter Varhol | JP0309PV_T |
| Deploy a Web Application with OC4J | David Gallardo | JP0309DG_T |
| Stay Flexible with Logic Scripts | Mark Nadelson | JP0309MN_T |
| Object Enterprise: The Promise of Java Everywhere | Peter Varhol | JP0309OE_T |
| Innovation Factory: JSIS and Source Code Dependencies | Claude Duguay | JP0309IF_T |
| Editor's Note: In the Mist | Kay Keppler | JP0309EN_T |
| Public Static: Tipping Point | Dan Ruby | JP0309PB_T |
| **August 2003 Vol. 7 No. 8** | | |
| Building Enterprise Portals | Howard Block, Rob Castle, and David Hritz | JP0308WP_T |
| The Readers Choose | Editors of *Java Pro* | JP0308RC_T |
| Architecting Security for Web Services | Mark O'Neill | JP0308MO_T |
| J2EE 1.4: A Web Services Kit | Peter Varhol | JP0308PV_T |
| Javatecture: The Trie of Knowledge | James W. Cooper | JP0308JT_T |
| Object Enterprise: Managing for Security | Peter Varhol | JP0308OE_T |
| Pro Shop: The Trouble with Distributed Objects | Daniel F. Savarese | JP0308PS_T |
| Editor's Note: The Next Big Thing | Kay Keppler | JP0308EN_T |
| Public Static: Center of Gravity | Dan Ruby | JP0308PB_T |
| Product Review: Rational XDE | Kito D. Mann | JP0308PR_T |
| **July 2003 Vol. 7 No. 7** | | |
| 7 Keys to Secure Java Software | Daniel F. Savarese | JP0307DS_T |
| Plug and Play with Java | Jason Byassee | JP0307JB_T |
| Visual Components: Employ Visual Text-Editing Features in JEditor | Claude Duguay | JP0307VC_T |
| Java Unplugged: Make Mobile Phones Smarter | Jeff Jurvis | JP0307JU_T |
| Pro Shop: Prove It, Don't Compute It | Daniel F. Savarese | JP0307PS_T |
| Editor's Note: Soup's On | Kay Keppler | JP0307EN_T |
| Public Static: The Permanent Evolution | Dan Ruby | JP0307PB_T |
| Product Review: SourceCafe | Curtis Krauskopf | JP0307PR_T |
| Book Reviews: *Core JSTL: Mastering the JSP Standard Tag Library* by David M. Geary and *Ant Developer's Handbook* by Alan Williamson et al. | Claude Duguay | JP0307BR_T |
| **June 2003 Vol. 7 No. 6** | | |
| Learning to Fly | Daniel F. Savarese | JP0306DS_T |
| Serialize Java Data Objects to XML | Charles D. Havener | JP0306CH_T |
| Make Applications Update-Aware | Mark Nadelson | JP0306MN_T |
| IXC Simplifies Xlet Communication | Xiaozhong Wang | JP0306XW_T |
| Pro Shop: Control Your Exceptions | Daniel F. Savarese | JP0306PS_T |
| Weblication: Java and the Model Driven Architecture | Peter Varhol | JP0306WC_T |
| Editor's Note: Where Have All the [Virtual] Flowers Gone? | Kay Keppler | JP0306EN_T |
| Public Static: Expanding the Pie | Dan Ruby | JP0306PB_T |
| **May 2003 Vol. 7 No. 5** | | |
| Integrated Testing Builds In Quality | Peter Varhol | JP0305PV_T |
| Streamline Your Exception Processing | Derek Ashmore | JP0305DA_T |
| Turn a JTable into a Spreadsheet | Thierry Manfé, | JP0305TM_T |
| Javatecture: A Ticket to Success | James W. Cooper | JP0305JT_T |
| Pro Shop: Write Tests to Refine Your Code | Daniel F. Savarese | JP0305PS_T |
| Editor's Note: It's the Real Thing | Kay Keppler | JP0305EN_T |
| Public Static: Bread and Chocolate | Dan Ruby | JP0305PB_T |
| Book Reviews: *Java Development with Ant* by Erik Hatcher & Steve Loughran, *Java Data Objects* by Robin M. Roos, and *Writers' Workshops & the Work of Making Things: Patterns, Poetry…* by Richard P. Gabriel | Claude Duguay James W. Cooper | JP0305BR_T |
| **April 2003 Vol. 7 No. 4** | | |
| Fail-Safe Interop for Your Enterprise | Daniel F. Savarese | JP0304DS_T |
| Monitor Multitiered Apps in One Location | Mark Nadelson | JP0304MN_T |
| Weblication: Putting a New Face on Web Interfaces | Peter Varhol | JP0304WC_T |
| Java To Go: Open a S.O.D.A. | Rick Grehan | JP0304JG_T |
| Pro Shop: On Optimizing Service Orchestration | Daniel F. Savarese | JP0304PS_T |
| Editor's Note: The Long and Winding Road | Kay Keppler | JP0304EN_T |
| Pulic Static: Groucho's Revenge | Dan Ruby | JP0304PB_T |
| Book Reviews: *JSTL in Action* by Shawn Bayern, *Sun ONE Studio Programming* by Rashim Mogha and Ruchi Bhargava, and *Programming LEGO Mindstorms with Java* by Giulio Ferrari et al. | Claude Duguay Rick Grehan | JP0304BR_T |
| **March 2003 Vol. 7 No. 3** | | |
| Get Small with Wireless Messaging and Mobile Media | Daniel F. Savarese | JP0303DS_T |
| Use Threading Tricks to Improve Programs | Osvaldo Pinali Doederlein | JP0303OD_T |
| MIDP 2.0 Is Ready for the Enterprise | David Hemphill and David G. Morehouse II | JP0303HM_T |
| Javatecture: Aspects, Concerns, and Java | James W. Cooper | JP0303JT_T |
| Visual Components: JIcon Dresses Up Your Interfaces | Claude Duguay | JP0303VC_T |
| Java Unplugged: The Push Is On | Jeff Jurvis | JP0303JU_T |
| Pro Shop: The Case for Conditional Compilation | Daniel F. Savarese | JP0303PS_T |
| Editor's Note: Flash! Here's Cold, Rational Fusion | Kay Keppler | JP0303EN_T |
| **February 2003 Vol. 7 No. 2** | | |
| Five Paths to Persistence | Daniel F. Savarese | JP0302DS_T |
| Build a Better Robot with Ant | Erik Hatcher | JP0302EH_T |
| Weblication: Automate Updates of Dynamic Web Apps | Peter Varhol | JP0302WC_T |
| Java To Go: Native PalmOS Databases | Rick Grehan | JP0302JG_T |
| Pro Shop: JAXB Revisited | Daniel F. Savarese | JP0302PS_T |
| Editor's Note: Skating Trials | Kay Keppler | JP0302EN_T |
| Public Static: Community Evolution | Dan Ruby | JP0302PB_T |
| **January 2003 Vol. 7 No. 1** | | |
| JMS Delivers the Message | Peter Varhol | JP0301PV_T |
| Just Browsing | Budi Kurniawan | JP0301BK_T |
| Create Rich Media Applications | Jonathon Maron and Jason Kinner | JP0301JM_T |
| Javatecture: Easy as Reeling Off a Log | James W. Cooper | JP0301JT_T |
| Visual Components: Line 'Em Up | Claude Duguay | JP0301VC_T |
| Pro Shop: Monitor Your Messages | Daniel F. Savarese | JP0301PS_T |
| Editor's Note: Securing the Franchise | Kay Keppler | JP0301EN_T |
| **December 2002 Vol. 6 No. 12** | | |
| J2ME Gets Personal | David Hemphill | JP0212DH_T |
| IDEs for Wireless Java | Rick Grehan | JP0212RG_T |
| Javatecture: The Focus Puller | James W. Cooper | JP0212JT_T |
| Weblication: Driving Java Development Through Testing | Peter Varhol | JP0212WC_T |
| Pro Shop: Eclipse vs. Swing | Daniel F. Savarese | JP0212PS_T |
| Editor's Note: From Server to Desktop | Kay Keppler | JP0212EN_T |
| Public Static: Method in Madness | Dan Ruby | JP0212PB_T |
| Product Review: JBuilder 7 Enterprise | Sue Spielman | JP0212PR_T |
| **November 2002 Vol. 6 No. 11** | | |
| Enterprise Software in a Services World | Peter Varhol | JP0211PV_T |
| 2002 *Java Pro* Technology Roundtable | Simon Phipps | JP0211TR_T |
| Java's Continuing Evolution | Daniel F. Savarese | JP0211DS_T |
| Javatecture: Handling SAX Errors | James W. Cooper | JP0211JT_T |
| Visual Components: A Window of Opportunity | Claude Duguay | JP0211VC_T |
| Java To Go: Direct to Palm | Rick Grehan | JP0211JG_T |
| Pro Shop: Close to Correct | Daniel F. Savarese | JP0211PS_T |
| Editor's Note: Blue Skies | Kay Keppler | JP0211EN_T |
| Public Static: Standard Deviation | Dan Ruby | JP0211PB_T |
| **October 2002 Vol. 6 No. 10** | | |
| What's New in EJB 2.1? | Tarak Modi | JP0210TM_T |
| Build a Smarter Search Engine | Baylor Wetzel | JP0210BW_T |
| Custom-Fit Web Development | Budi Kurniawan | JP0210BK_T |
| JAXR: A Web Services Building Block | Sameer Tyagi | JP0210ST_T |
| Weblication: Profile Your Web Services | Peter Varhol | JP0210WC_T |
| Javatecture: Practicing Safer SAX | James W. Cooper | JP0210JT_T |
| Visual Components: Just What You Want to See | Claude Duguay | JP0210VC_T |
| Java To Go: Liten Up | Rick Grehan | JP0210JG_T |
| Pro Shop: How Hot Is HotSpot? | Daniel F. Savarese | JP0210PS_T |
| Editor's Note: The Energy Equation | Kay Keppler | JP0210EN_T |
| Public Static: Whom Can You Trust? | Dan Ruby | JP0210DR_T |
| **September 2002 Vol. 6 No. 9** | | |
| Extending the Enterprise | Peter Varhol | WE0201PV_T |
| Nokia's Data Strategy | James E. Fawcette | WE0201JF_T |
| Deliver Big Functionality on Small devices | Rick Grehan | WE0201RG_T |
| Execute Applications Remotely in Java | Mark Nadelson | JP0209MN_T |
| Hot Java Devices | Lee Sherman | WE0201LS_T |
| Weblication: Java to XML and Back Again | Peter Varhol | JP0209WC_T |
| Javatecture: Stating Your Preference | James W. Cooper | JP0209JT_T |
| Visual Components: Keep Tabs on Your Software | Claude Duguay | JP0209VC_T |
| Java To Go: A Catalog Model | Rick Grehan | JP0209JG_T |
| Pro Shop: Application, Heal Thyself | Daniel F. Savarese | JP0209PS_T |
| Editor's Note: Buckle Up | Kay Keppler | JP0209EN_T |
| Public Static: Casualty of War | Dan Ruby | JP0209PB_T |
| **August 2002 Vol. 6 No. 8** | | |
| Undaunted Testing | Mark Nadelson and Marina Evenstein | JP0208MN_T |
| JDBC: The Next Generation | Brett Spell | JP0208BS_T |
| A View from a JTree | Budi Kurniawan | JP0208BK_T |
| A New Security Blanket | Josh Street | JP0208JS_T |
| Weblication: Build Quality Apps with JMeter | Peter Varhol | JP0208WC_T |
| Javatecture: Native Intelligence | James W. Cooper | JP0208JT_T |
| Visual Components: An Easy Way to Sort Things Out | Claude Duguay | JP0208VC_T |
| Java To Go: Setting a Record | Rick Grehan | JP0208JG_T |
| Pro Shop: Whatever Happened to Jini? | Daniel F. Savarese | JP0208PS_T |
| Editor's Note: A Free Lunch | Kay Keppler | JP0208EN_T |
| Public Static: Nearly Full Monty | Dan Ruby | JP0208PB_T |

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 Java EE 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!



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

# 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

## Advertiser Index

# The Two Sides of Progress

Guest Opinion
by Onno **KLUYT**

A lot is being said and written about standards and innovation in technology: standards are a roadblock to innovation because the process of standardization is too slow to capture innovation in a timely manner, and standards are more about politics than technology. In fact, across the industry there is plenty of evidence of how standards and innovation work together to *advance* technology. Being closest to the Java Community Process (JCP), I know that this community has accomplished a marvelous thing: redefining standards and innovation as the sides of the same process—progress. Let's look at a few Java Specification Requests (JSRs) that I encourage you to check out for yourself at the technical sessions (TS), hands-on labs (LAB), and birds-of-a-feather (BOF) sessions at the 2006 JavaOne Conference in San Francisco.

Let's begin with JSR 245, JavaServer Pages (JSP), and JSR 252, JavaServer Faces (JSF) 1.2 (LAB-4255 and BOF-2311). JSR 245 is the next revision of the JSP specification, and it improves alignment with JSF and enhances ease of development. Similarly, JSR 252 updates the 1.1 version of the JSF specification. These JSRs provide good examples of how standards build on standards and prepare and inspire innovation. They also demonstrate how innovation is worked into platform standards as the so-called *umbrella JSRs*, including JSR 244, Java EE 5.

For another example of advances at the standards-innovation intersection, there's JSR 224, JAX-WS (TS-1194). The major focus of this standard is ease of development to allow the technology to be used by a wide circle of developers and simplify their tasks. The specification extends JAX-RPC in a number of ways including alignment with JSR 181, Web Services Metadata for the Java Platform. You'll also find out how the spec strongly aligns with JSR 222, Java Architecture for XML Binding (JAXB) 2.0 (TS-1607), to which it delegates all data binding-related tasks and how it supports new versions of external standards from organizations such as W3C and WS-I.

Another JSR-based session for JSR 286, Portlet Specification 2.0 (TS-3627) and headed by IBM, showcases the functionality that will be added to the new portlet specifications. This API advances and will be binary compatible with version 1.0 defined in JSR 168. If you want to find out how a standard developed today seeds innovation, attend the Java Module System (BOF-0684) session that is targeted to be delivered as a component of Java SE 7.0 ("Dolphin"). The Sun-developed specification sets out to define a distribution format and a repository for collections of Java code and related resources as well as discovery, loading, and integrity mechanisms at runtime.

JSR 220, EJB 3.0, is another example of how standards and innovation find a way to build on the strengths co-leads Sun and Oracle bring to the table and as a result cause progress to happen. One of the results this JSR has yielded is a simplified persistence architecture. Key features of the Java Persistence API (TS-3395) will be highlighted, including those introduced since the publication of the JSR 220 public draft.

Platform JSRs like JSR 270, Java SE 6 Mustang Release Contents, are a special case of close interplay between standards and innovation. An umbrella JSR builds on the sum of innovations provided by the so-called *point* JSRs, and scripting features in Java SE 6, including the scripting APIs and the JavaScript ScriptEngine, will be presented (TS-1382).

IBM and BEA, co-leads of JSR 235, Service Data Objects (SDO), will present how developers will be able to simplify data access and representation in service-oriented software by replacing data access models with a uniform abstraction for creating, retrieving, updating, and deleting business data used by service implementations (TS-3676). The SDO specification currently under development standardizes data objects in terms of change history, compound data objects, dynamic and generated APIs, metadata, support for XML and Web services, neutral representation of business data, import/export from common formats, validation and constraints, relationship integrity, and navigation.

Java ME is where perhaps the most obvious advances fueled by community-driven standardization and innovation is occurring. Two spec leads from Nokia team up to present the recently finalized JSR 256, Mobile Sensor API (BOF-2810), which defines basic sensor functionality for mobile devices and extends the usability and choice of sensors for Java ME applications. A perfect example of interplay between standardization and innovation is JSR 248, Mobile Service Architecture (TS-4936), developed by Nokia and Vodafone, that creates a mobile service architecture and platform definition for high-volume wireless handsets. JSR 232, Mobile Operational Management (TS-3757), led by Motorola and Nokia, is the topic of an introduction to this spec under development and the benefits it sets out to pass on to developers. And co-leads Nokia and Motorola of JSR 272, Mobile Broadcast Service API for Handheld Terminals (TS-4693), will present the features of this set of APIs.

These are just a handful of examples of the symbiotic nature of innovation and standards. At the JavaOne conference you'll be able to see this system in action. Also visit the JCP.org (http://JCP.org) to get the latest on innovating Java technology. JP

Onno Kluyt is chair of the JCP (http://JCP.org).

# Innovations by InterSystems


## Real-time data analytics with a real-fast database.

Imagine being able to query a lightning-fast operational database in real time.

Now you can, with our multidimensional database for transaction processing and real-time analytics.

Only Caché combines robust objects and robust SQL, thus eliminating object-relational mapping. It requires little administration, delivers speed and scalability on minimal hardware, and comes with a rapid application development environment.

These innovations mean faster time-to-market, lower cost of operations, and higher application performance. We back these claims with this money-back guarantee: *Buy Caché for new application development, and for up to one year you can return your license for a full refund if you are unhappy for any reason.**

Innovative database. Guaranteed performance.

InterSystems CACHÉ™

## Rapid integration platform makes applications perform together.

Imagine being able to get your applications to perform together as an ensemble. Easily.

Now you can, with our universal integration platform.

Ensemble is the first fusion of an integration server, data server, application server, and portal development software – in a single, seamless product. This is the complete ensemble of technologies needed for rapid integration, fast development, and easy management.

These innovations mean all of your integration projects will be completed on time and on budget, with a simplified learning curve for your IT staff. We back these claims with this money-back guarantee: *For up to one year after you purchase Ensemble, if you are unhappy for any reason, we'll refund 100% of your license fee.**

Innovative integration. Guaranteed performance.

InterSystems ENSEMBLE™

For a free copy of CACHÉ, or to request a free ENSEMBLE proof-of-concept project, visit **www.InterSystems.com/Free8Q**

*Read about our money-back guarantees at the web page shown above.