JAVA DEVELOPER'S JOURNAL

The World's Leading Java Resource

June 2001 Volume: 6 Issue: 6

From the Editor
by Alan Williamson pg. 7

Industry Commentaries
by Richard Green pg. 8
by Bruce Scott pg. 146

Product Reviews
Sun Blade pg. 20
WebLogic 6.0 pg.
ServletExec pg.
iPlanet pg. 6

J2EE Editorial
by Ajit Sagar pg. 22

Interview
Larry Freeman, Manager
J2EE Blueprints Team pg. 26

First Look
Optimal J pg. 140

J2ME Editorial
by Jason Briggs pg. 144

EXCLUSIVE: INTERVIEW WITH JDJ EDGE conference&expo KEYNOTER...SCOTT DIETZEN

JDJ 2.0 SPECIAL FOCUS ON J2EE, J2SE & J2ME

J2EE: The Challenges and Pitfalls of J2EE — Chris Kampmeier 36
J2EE: Growing JSP and Servlet Sites — Patrick Sean Neville 44
J2EE: WebSphere Column — Brady Flowers 56
J2EE: The Impact of EJB 2.0 — Tyler Jewell 64
J2SE: Building a Telephone/Voice Portal — Klinner & Walker 78
J2SE: Making a Mountain out of an Anthill — Neal Ford 86
J2SE: JNI Programming in C/C+ — Blair Wyman 96
J2SE: Evolving Functionality — Satya Komatineni 104
J2SE: Optimization and Java — Irvin J. Lustig 110
J2SE: Using JavaBeans in Dreamweaver — Barbarelli & Deming 118
J2SE: Deployment of JavaServer Pages — John Goodson 126
J2SE: Java Quirks – Dates and Calendars — Roedy Green 134
J2ME: The Incredibly Shrinking Platform — Jason Briggs 148
J2ME: A UI Framework for the MIDP API — Glen Cordrey 154
J2ME: Java Thick Clients with J2ME — Glenn Coates 164

ALAN WILLIAMSON, EDITOR-IN-CHIEF

The Dawn of Content

It has been one hell of a journey to get where we are today. What you're holding in your hands is the first of the new breed of ***Java Developer's Journal***. On behalf of the team here, allow me to introduce ***JDJ*** 2.0.

This has been the accumulation of four months' preparation, and we hope you like the new look and feel of your beloved ***JDJ***. We aren't quite settled in yet, as this issue is the crowning glory of the year, with JavaOne, and it kind of throws out the normal rules of engagement.

One of the first changes you'll notice is the three major sections of the magazine that reflect the three editions of Java 2. We'll be focusing on each edition with equal journalistic flair. There's a lot going on in the Java space and we want to bring you the best of the best each and every month.

As you know, the Java spectrum is simply too wide for one single person to effectively report the latest goings-on. To that end we have invited people who are renowned in their respective fields to become section editors. These editors are responsible for editing and sourcing the content for each section. You would think it makes my job easier, but sadly it doesn't seem to work out quite like that. If anyone can figure that out, let me know!

Ajit Sagar has kindly accepted the role of J2EE editor and, if his work for this issue is anything to go by, our J2EE section is in great hands. I look forward to working with Ajit in subsequent issues as we bring you cutting-edge, J2EE-focused articles.

For our J2ME section I feel fortunate in snagging a great writer and authority on this new emerging field. Jason Briggs joins us in earnest in the next issue, and I know that we here at ***JDJ*** are looking forward to working with him.

At the time of writing we haven't made a firm decision on our J2SE editor. Choosing an editor is something we don't take lightly; however, I am interviewing a number of people at JavaOne, so I'll let you know next month.

In addition to bringing you richer content, we're going to be running a series of beginner's features to ease the learning curve that many of you experience when starting out on the Java road. We're looking to make the transition easier for our beginners and to help nurture them into world-class developers. We've lined up some great writers for this series and I look forward to introducing this feature to you next month.

As editor-in-chief I have a responsibility to set the tone of the magazine and ensure continuity each month. Those of you familiar with my writings know I'm passionate about a number of areas of Java and that I'm not scared to whip up a debate about any of the subjects. However, there is one area that I wish to champion – the overall quality of the skill base of our new recruits to the Java community. With the help of our section editors and Advisory Panel we'll be delivering articles for all skill levels, from beginner right up to top-notch developer.

As I pack my kilt in preparation for JavaOne, I hope I get to meet and talk to a lot of you while I'm out there. If you see a strange-looking bugger in a skirt, be sure to stop him; chances are it's me! If it's not, please apologize and quickly move on. I'm more than happy to talk to you about what we're up to here at ***JDJ*** and welcome your feedback.

I look forward to serving you in the future… ◆

alan@sys-con.com

AUTHOR BIO

Alan Williamson is editor-in-chief of Java Developer's Journal. *In his spare time he holds the reins at n-ary (consulting) Ltd, one of the first companies in the UK to specialize in Java at the server side. Reach him at alan@n-ary.com (www.n-ary.com) and rumor has it he welcomes all suggestions and comments!*

WRITTEN BY RICHARD GREEN

Train Once, Write Anywhere

Since its introduction in 1995 Java technology has been known for its "cross-platform" compatibility. This ability to run applications on multiple platforms made Java big news and presented developers with a new and unique application development platform. In the beginning cross-platform meant that Java applications could run on several distinct workstation-system platforms such as Windows, Macintosh, and UNIX.

In 1999, we announced that the Java platform would extend itself upward into the enterprise server with the Enterprise Edition, and downward into the client-device space with the Micro Edition. With the release of J2EE in December 1999, cross-platform began to get a new, vertical meaning since compatibility extended now from the desktop to the server. This compatibility has formed the basis for J2EE's incredible success as the standard for networked-enterprise application development, especially in the application server space.

Today, we're starting to see yet another manifestation of cross-platform as J2ME-enabled consumer electronics devices come to market. Now Java spans vertically from the server to the desktop to the modern client. However, we also introduce a new meaning to cross-platform because J2ME spans horizontally across all the new client devices such as cell phones, pagers, PDAs, TV set-top boxes, automotive systems, home gateways, games, and even smartcards. You might call this "cross-client" compatibility.

When you think about it, this is an awesome shift and has real impact on the meaning of the term *developer*. For the industry, putting Java into cell phones, like those introduced in late January by NTT DoCoMo in Japan or the Motorola phones for the Nextel network that debuted in April, means the digital convergence we've been talking about for the last decade is really happening. For developers it means the addressable market for their work has increased dramatically since industry analysts predict that billions of devices will log on to the Internet and other networks like DoCoMo's and Nextel's.

When ***JDJ***'s editor-in-chief, Alan Williamson, visited me this spring, we had an interesting discussion about just what a developer is. Alan posited that the new devices that run Java would appeal to the veteran developers, you know, the guys who write close to the silicon and can keep track of memory allocation and the other complexities that come with writing programs for a constrained environment like a 640KB cell phone.

I countered that now there are, in effect, two categories of developers: the traditional that Alan described and a new kind who are more focused on content than logic. The latter, of course, is not as technical as the traditional developer. They worry less about memory size and more about page size. They're game, map-application, and e-commerce shopping page developers, or online newspaper editors, but I think they do what the traditional developer did at the highest level: provide people with information they need to get their job done or their life arranged.

Developers need to understand that the network has not only changed the paradigm for building and running applications, it's changing the developer paradigm too. In the world of Java there are many implications. Who is a developer today? Is it only someone who can craft an EJB, or does it include a servlet developer or even a JSP developer who's "developing" a Web page that's fundamentally content?

For years developers built on the platform they targeted. Today, as the IT industry crosses the great divide from addressing just traditional computers to addressing consumer electronics devices, it becomes impossible to develop to a single hardware platform. The consumer space makes the desktop look like the height of homogeneity. In the consumer electronics world there are dozens, if not hundreds, of different system types and many of them are completely proprietary. With Java supporting applications and services horizontally on all the different client-device types, as well as vertically from the client to the server, developers can build services that will reach millions of devices in the near term, and billions in the long term if the analysts are correct. That's a lot of sophisticated, increasingly intelligent applications and services that will be developed and targeted at a massive number of users.

Ultimately, consumers will have a wide variety of devices to access their services and will use the one most convenient at any given time. At work they might use a desktop computer; at home, a PC or television set-top box; while traveling, they can use an automotive system; and in between, their cell phone. This means developers will have to keep in mind the needs of all these devices when developing networked applications and services.

We, of course, feel that Java is the only development and deployment environment that can scale vertically and horizontally. This capability is crucial for the fast-emerging era of smart networked services, but it also provides a huge benefit for developers because once they know the Java programming language and have experience with how the Java platform deals with different system types, the reach and range of their skills becomes almost unlimited. I call this TOWA: train once, write anywhere. Java developers – regardless of whether they're hard-core application developers or the new breed of content developers – can take their skills almost anywhere in the digital domain they care to go.

richard.green@sun.com

AUTHOR BIO

Richard Green is vice president of Java Software Development for Sun Microsystems, Inc. He drives the strategy, product development, support, and compliance technologies for all of Sun's Java/Jini/XML platforms. Richard holds a BA/MA from the State University of New York at Albany.

SPECIAL SECTION: JAVA FAQ

JDJ recognizes that not all our readers are world-class developers, in fact many of you don't even develop. To this end we'd like to introduce a regular column for those of you that are either just starting out or simply want to get a feel for what's happening out there in Java land. If your question isn't answered here, feel free to ask any of our editors and we'll see if we can help you out.

What is Java? Is it a product?

When Java first appeared on the public radar in 1995, it was heralded as the second coming. To that end the message conveyed by the marketers was at times mixed, and more often than not, wrong. Java was not the answer to everything, it was merely a new computer language that attempted to address some of the problems that previous languages failed to answer or found difficult to solve. It was not a product. It was not something you could buy from your local computer retailer. Java was not the solution, merely a tool.

Never before in the history of languages has one language captured the imagination of the press the way Java has. The reason for this is a little unclear but, as a result, more confusion has built up around Java than any other language. Before long board rooms wanted "a Java solution" before really understanding what it was. When Java first came out, it was previewed within the Netscape (and then IE) browser. It became known as the "little gray rectangle" and, due to early implementations, had to fight the backlash that it was apparently slow. Since 1995, Java has been doing a lot of growing up and this early criticism of speed is no longer applicable.

Who invented Java and why? What was wrong with the other languages available?

You could write a book about the history of Java, but the short-short version is as follows. Around 1990, James Gosling, Patrick Naughton, and Mike Sheridan got together to work on a project – to come up with an alternative to C++ that would address some of the issues associated with this language. At that time all three were working at Sun and, after an initial demonstration of their project to Bill Joy and Scott McNealy, it was decided to spin off a separate company to see if this "Oak" language could be successful in providing Time-Warner with a set-top box operating system. Fortunately, as history dictates, it failed. The project was brought back under the wing of Sun, where after a weekend of legendary coding by Naughton and later additional development by Jonathon Payne, HotJava was born, the first Java-based Web browser.

From there on in the popularity of Java simply grew, capturing the imagination of the development community. The main reason it was an instant hit was its "WORA" claim – Write Once, Run Anywhere. This allowed a developer to develop a single application once and deploy it to many platforms without recompiling/redeveloping a single line of code.

What is this "JVM" acronym I keep seeing around?

Following the answer from the previous question, Java's ability to run anywhere is a result of the way it's designed. When you develop a program in Java, you don't compile to the native platform but to a virtual computer. This virtual computer is where your program will run. When you compile your Java program it's converted to bytecode. It's this bytecode that you can run within the virtual computer. The virtual computer, of course, is so called because it doesn't physically exist. Therefore, a piece of software plays the role of the virtual computer, which is more commonly known as the Java Virtual Machine (JVM). The JVM is a native, specific piece of software that runs on top of a given platform. The Java code then runs on top of that. For example, the JVM for a Solaris machine is different than the JVM for the MS-Windows architecture, but the functionality they provide is the same. This is the key to providing a WORA environment.

Does JavaScript have anything to do with Java?

No. JavaScript is a language developed by Netscape. Originally it was designed to have a closer relationship; this never materialized but the name stuck. It has created no end of confusion and headaches for all those who don't know the difference.

What's all the fuss about with respect to Microsoft and Sun?

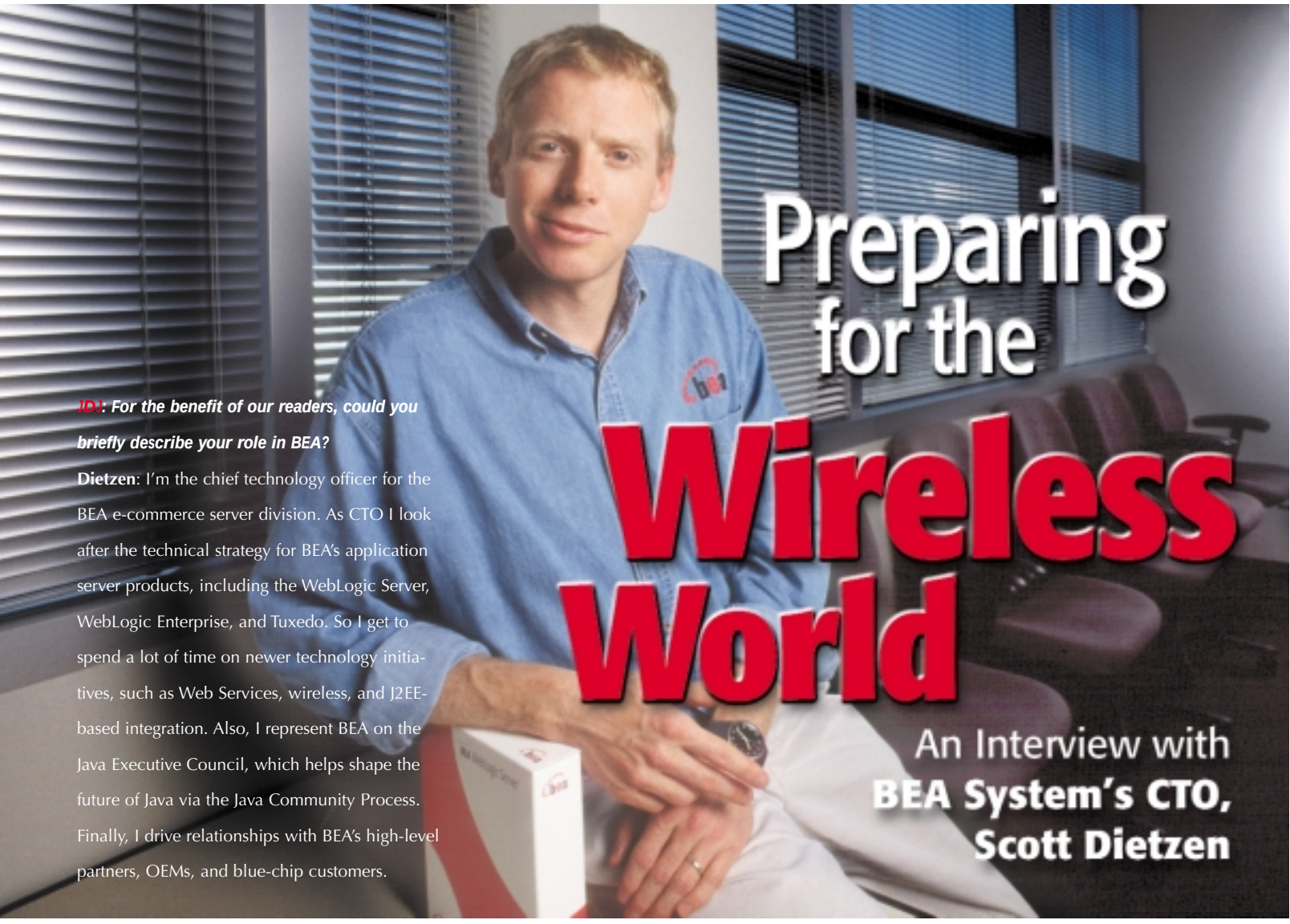
'Tis a complicated story, but in layman's terms: when Sun licensed the Java source code to Microsoft to allow them to build a JVM for the Windows platform, Microsoft modified it a little to have greater access to some of the Windows-specific components. This modification broke the WORA philosophy and to that end, Sun felt it was no longer Java and wanted Microsoft to restore their JVM to be 100% compliant. Microsoft didn't feel the same. Hence the court case.

Intenational Advisory Board

www.sys-con.com

c Spread

gistics.com



Preparing for the Wireless World

An Interview with BEA System's CTO, Scott Dietzen

JDJ: For the benefit of our readers, could you briefly describe your role in BEA?

Dietzen: I'm the chief technology officer for the BEA e-commerce server division. As CTO I look after the technical strategy for BEA's application server products, including the WebLogic Server, WebLogic Enterprise, and Tuxedo. So I get to spend a lot of time on newer technology initiatives, such as Web Services, wireless, and J2EE-based integration. Also, I represent BEA on the Java Executive Council, which helps shape the future of Java via the Java Community Process. Finally, I drive relationships with BEA's high-level partners, OEMs, and blue-chip customers.

INTERVIEWED BY
AJIT SAGAR

ajit@sys-con.com

JDJ: BEA seems to be everywhere on the application server map. Can you tell us some of the plans you have for expansion in the next couple of years?

Dietzen: Application servers are growing up into what BEA has been calling an e-business platform. Much of this new functionality is being packaged directly within the application server, such as support for multichannel applications – a unifying architecture for delivering information services across Web, wireless, voice, and Web Services. With WebLogic Server 6.1, we're also automating the generation of Web Services bindings for existing J2EE applications by generating SOAP and WSDL from EJB session beans and vice versa. We've substantially upgraded JMS performance and are now supporting IP multicasting.

But e-business platforms are more than application servers. I'm most excited right now about the impact Java and Web Services standards are having on integration. To date, the EAI market has failed to deliver on its promise, because (1) the frameworks have been built on proprietary middleware platforms with no standard data model, and (2) adapters to the legacy systems need to be "one off'ed" for each EAI solution. So (1) is now being addressed by J2EE, XML, and Web Services, while (2) is being taken care of by the J2EE Connector Architecture (JCA). BEA believes this one–two punch will transform the integration market the same way J2EE transformed application servers.

We're also growing WebLogic up toward the line of business with Personalization, Portal, Commerce, and Campaign Manager. These products empower business users to aggregate and tailor information services to better meet the needs of their customers and business partners.

JDJ: I attended your BEA User's Group Conference about a month ago. It seems that you're close to completing the story on a true e-business platform. How do you plan to distinguish yourself from other leading vendors like iPlanet, WebSphere, and ATG?

Dietzen: Of course, competition is good for the industry and good for BEA. Longer term, the biggest advantage that the Java community offers over Microsoft .NET may be the breadth of innovation happening on our platform. Our second rule is that most competition is really "coopitition." Sun remains one of our most strategic partners even though we compete with iPlanet. And while WebSphere remains our primary competitor, BEA complements more than 99% of IBM's business. Even ATG is becoming complementary. I was excited to hear that ATG was following the direction of BroadVision and Vignette in replacing their proprietary middleware with a J2EE application server. So how is BEA maintaining the #1 position against strong competition?

- ***Reference customers:*** We're leading the J2EE market in production deployment.
- ***Alliances:*** BEA has over 1,600 value-added partners, including almost all of the enterprise software vendors (PeopleSoft, BroadVision, Vignette, Ariba, and so on), and virtually all of the leading systems integrators (Accenture, CSC, KPMG, and so on). This is the most critical litmus test for a platform: How many other vendors are standing on top of you?
- ***Technology leadership:*** WebLogic's initial claim to fame came out of getting new Java technologies to market as early as possible. We're continuing that trend with EJB 2.0, JCA, and transparency for Web Services.
- ***Independence:*** The final point I would cite is BEA's independence. Unlike our larger competitors, we don't have a vested interest in hardware, operating systems, databases, or professional services. That means ISVs and end users can partner with BEA knowing that their applications can be deployed on Solaris, Linux, Windows, HP-UX, AIX, AS/400, and even OS/390, and deployed with Oracle, DB2, MS SQL Server, Informix, Sybase, and so on.

JDJ: Whom do you consider your strongest competition?

Dietzen: IBM is the only competitor we see on a regular basis. In our most recent earnings report, you'll see the latest evidence that we're growing our WebLogic business faster than IBM WebSphere is growing, and we're growing from a substantially larger base.

JDJ: Could you broadly categorize your product line?

Dietzen: Our product line makes up the WebLogic E-Business Platform that provides the essential infrastructure for building an integrated e-business. The platform includes the market-leading WebLogic application servers plus commerce, personalization, and portal components for rapidly developing e-commerce applications to win and retain customers. It also includes newly updated integration capabilities that support new ways of automating business processes, and linking and interacting with suppliers, partners, and existing deployed applications.

JDJ: One of the things I took away from the conference was that although you have a broad range of offerings, there's still some integration work to be done. For example, most of the products run on WebLogic 5.1 instead of 6.0. How will you bridge the gap?

Dietzen: Well, the good news is that all of our WebLogic products now run on WebLogic Server 6.0. The time frame you refer to was several months ago, shortly after WebLogic Server 6.0 was introduced. The only alternative to this gap would be our holding up the release of WebLogic Server until our value-added e-commerce and integration products were certified on the new platform. Given the continuing demand for new application server technology, we felt this was the wrong thing to do.

JDJ: Who are some of your biggest customers?

Dietzen: Some of the big names are FedEx, J.P. Morgan Chase, Charles Schwab, Nokia, DHL, United, Delta, British Airways, General Electric, Verizon, Amazon, and NTT DoCoMo. All told, we have over 9,400 corporate customers, including 100% of the Fortune Global 500 financial services companies, telcos, computer/office equipment manufacturers, pharmaceuticals, airlines, and delivery services.

JDJ: This year it seems that the application server vendors are continuing their move toward being "one-stop-shops" for all B2B2C frameworks. BEA seems to be on the same bandwagon. What would you say is unique about what your company offers?

Dietzen: In addition to what I mentioned earlier about key differentiators for BEA, I would say our platform is unique in how it offers a comprehensive e-business platform for building and deploying e-business applications, spanning the full spectrum from personalized e-commerce and customer relationship management on the front end to total e-business integration on the back end, including integration across the enterprise and outside the enterprise to the supply chain and other business partners. We're able to provide all of this on the industry's only fully integrated platform based completely on open industry standards, such as XML, Web Services, and J2EE.

JDJ: In which areas do you think WebLogic needs more enhancements to compete in the market? For example, I think you need more work on the presentation end.

Dietzen: I'd agree with you if the point regards multichannel presentation services. BEA's current offering around the Web is very powerful, incorporating graphical tools and content management solutions from our strategic partners Macromedia, WebGain, Documentum, Interwoven, and Vignette. However, the multichannel problem is more difficult. While there's a lot of marketing around wireless technology solutions, the industry hasn't yet solved all of the challenges. For example, how can I automatically rerender my content for the diversity of wireless devices – different screen sizes, different browsers and markup languages, and even different interaction models? We've got some very large wireless deployments under our belt, but no one has yet made wireless as easy as the Web is today.

Overall, while I remain well satisfied with BEA's ability to compete, we also must continue to aggressively improve our platform to keep our lead – J2EE-based application integration; business Web services for complex, multiparty, transactional applications; broadly distributed caching and replication; opening up J2EE applications to the line of business manager for access to real-time operational data; and working with our partners on higher-level tools for J2EE application development. These are all areas that we're investing in aggressively.

JDJ: With consolidations and mergers, where do you think the application server market is going? Where does BEA stand in that market?

Dietzen: We believe the J2EE application server represents the predominant platform for e-business applications. For BEA to do well, Java and J2EE must continue to do well. At present, the only emerging alternative appears to be Microsoft's .NET. As .NET continues to flesh out over the coming year, I expect to see increasing and healthy competition between the Java and Microsoft stacks. We all need to help educate the market that the real game is Java and EJB versus C# and COM+, while Web Services and XML are features of both platforms. Indeed, the adoption of the common Web Services stack – SOAP, WSDL, UDDI – offers the Java community the promise of "out-of-the-box" interoperability with Microsoft.

Much as Microsoft and Oracle emerged as the predominant platform for the client/server generation, BEA aspires to offer the server-side platform for "networked" appli-

"We believe the J2EE application server represents the **predominant platform for e-business applications**"

cations or "Web apps." This is a large and growing market, but consolidation is going to continue. The costs associated with delivering a comprehensive e- business platform have doubled over the past couple of years as more and more functions – personalization, clustering, integration, Web Services, wireless, and more – have become built in. The industry will seek to amortize these costs by standardizing on the platforms that are the richest and most ubiquitous. I believe that to date BEA is benefiting most from this consolidation.

JDJ: Does BEA plan to get into actual application design; that is, do you plan to step into industry verticals or do you plan to always be application enablers? Do you plan to continue expansion in horizontal technology offerings?

Dietzen: We'll stay focused on e-business infrastructure. BEA doesn't have the expertise to compete in vertical applications. More important, to do so would alienate our most critical allies – the independent software vendors.

For application design, we do offer professional services to our strategic customers, but that's most often used to validate their use of BEA technologies. We couldn't possibly hope to grow our own professional services organization as fast as the demand. Leveraging our system-integration partners is the only scalable approach.

JDJ: With Commerce Server and Collaborate, aren't you stepping into the same space as some of the B2B solution providers? For example, the companies who build electronic marketplaces? Is this intentional?

Dietzen: Actually, I don't quite buy this. For customers looking for an "out-of-the-box" B2B solution, they should absolutely continue to go to our ISV partners. However, other companies are looking to differentiate via home-built B2B solutions. So this generally comes down to a buy versus build analysis. Provided the underlying platform is J2EE and WebLogic, both approaches offer superior scalability, reliability, integration, and investment protection. So getting the customer to choose a WebLogic and J2EE foundation is BEA's goal.

JDJ: When do you plan to integrate WebLogic Server with Tuxedo? Can you briefly outline for our readers the advantages of such an enterprise environment?

Dietzen: We have very tight integration between WebLogic and Tuxedo today, but don't look for these to become a single product that runs in one address space. The things that make Tuxedo the world's best application server for C and C++, don't work well with Java. BEA is offering our enterprise customers the marriage of the two market leaders – WebLogic Server is the leading Java/Web container, while Tuxedo is the market's leading container for C/C++ (Tux even supports COBOL).

Far more important than merging WebLogic and Tuxedo is tight interoperability between the two. The most recent release of Tuxedo allows direct invocation from Tuxedo to WebLogic EJBs (via IIOP). Of course, we go in the opposite direction as well, allowing Java applications to easily invoke CORBA objects or Tuxedo services. All with security and transaction sharing between the containers.

JDJ: WebLogic 6.0 is a big release for BEA. Would you like to describe the main enhancements as compared to Release 5.1? How many of your current customers have made the switch?

Dietzen: WebLogic Server 6.0 is the platform of choice for nearly all our customers' new applications. In 6.0 we introduced a powerful new management framework built on JMX with a Web-based console, a new scalable, high performance messaging (JMS) engine, major transaction manager enhancements built in Java by the Tuxedo team, along with numerous improvements in installation, ease of use, and in XML.

Now we've gone one better with WebLogic Server 6.1, available at JavaOne. 6.1 adds Web Services automation, J2EE updates (EJB 2.0, JCA, JSP 1.2, Servlet 2.3), and support for caching and replicating "read mostly" entity beans efficiently across a large cluster. Of course, we also have a substantial deployed base of 4.5 and 5.1 users that we'll continue to look after.

JDJ: It seems that all the application server vendors are offering products in the process management space. BEA has entered this space with WebLogic Process Integrator and Collaborate. Do you think that with these environments, the way business applications are designed will change?

Dietzen: We believe integration needs to be designed in from the beginning. Adding these capabilities to the WebLogic platform, along with the Web Services, JCA, JMS and other integration enablers, will enable our customers to build "integration-ready" applications.

JDJ: How well do you integrate with back-end systems? Besides JCA, do you have any partners that you integrate with directly? For example, WebMethods or even BizTalk?

Dietzen: I think that's already happening. For quite some time people have been talking about workflow automation tools, but still hand coding such tasks. But by building applications on J2EE, you're already building in the integration points (EJB, JMS) that a Java-based workflow solution plugs into. The same thing is true for integration – by building on WebLogic you inherit an integration platform for tapping into legacy systems. That allows our ISVs to offer products that are "built to integrate" without having to invest in the integration technology.

JDJ: Do you plan to enhance support for Microsoft environments in the future, or do you plan to continue in a pure Java arena?

Dietzen: Both. We've been committed to integration with the Microsoft platform all along and will continue to do so going forward. We've been providing native integration with IIS (MS Web Server), COM, SQL Server, and the NT registry for several years, and with Web Services we have the promise of "out-of-the-box" interoperation with the Microsoft environment. However, we have no plans to embrace application development in C#. Instead, BEA will continue to ensure that our Java/J2EE-based products work well on the Microsoft platform.

JDJ: How do you plan to address enterprise-level concerns like scalability, robustness, and security?

Dietzen: We think we've done a unique and good job of this already. You might say it's BEA's raison d'etre. BEA software is now running sites with more than 10,000 transactions per second. We've scaled a single Java application to run across a 60-server/240 processor cluster. We're offering Web and EJB session protection that doesn't rely on a single process pair (scaling bottleneck) or writing updates to a DBMS (far too expensive). Count on BEA to do our best to maintain our competitive advantage here.

JDJ: How far have you advanced in the wireless and mobile market? Where do you think this market is going?

Dietzen: We're providing customers with a single platform that they can use for multichannel applications – wireless and wireline, as well as voice and Web Services. We expect the wireless application market will be very large in a couple of years. Today, the growth is uneven. Northern Europe, Japan, and some other Asian marketplaces are well ahead of the U.S. While I'd argue that this market is not quite mainstream yet, we have many customers that have deployed first-generation wireless applications already.

JDJ: How does XML fit into your overall technology blueprint?

Dietzen: XML has become a fundamental part of the application server, and we've been supporting XML for quite some time. XML clearly plays a big role in Web Services, as well as the other integration services we have built into the overall WebLogic platform.

JDJ: For Java developers who are not familiar with BEA products, what would be a good place to start incorporating them into their existing applications?

Dietzen: Certainly, anyone building or planning to build Web, wireless, or Web Services applications in Java should look at WebLogic as well as our competitors. The sooner the better, since I'm convinced that virtually all server-side Java applications are going to be deployed on application servers for ease of development, manageability, security, quality of service, and so on.

Sun Blade

by Sun Microsystems

Is it that sharp?

REVIEWED BY ALAN WILLIAMSON

alan@sys-con.com

When Sun Microsystems rings you up and asks whether or not you'd be interested in playing with a new desktop system they've just announced that's aimed at Java developers, amongst many, what can you do but nod affirmatively. After much paperwork to allow the machine to leave the shores of the U.S., it arrived here in Scotland where it was immediately unpacked and fired up. That was two months ago.

So how did I get on? Well, let me take you through the trials and tribulations of working with a Sun desktop machine, as opposed to a Windows-based system, to develop Java applications.

The first thing I noticed about the physical box is that it's undeniably drop-dead gorgeous. Traditionally Apple computers have always triumphed in this area over and above the dull, lifeless PC boxes. Sit an Apple beside a PC and each time the Apple beckons you closer. Sit an Apple beside the Blade system and suddenly the Apple has serious competition on its hands in the looks department. Increasingly, hardware manufacturers are moving away from "plain-Jane" boxes to the more eye-pleasing sexier packaging. So hurrah on that score.

I asked to have the Forte development tools preinstalled on the machine so I could start building as soon as it was powered up. As you can see from the pullout, the specification of the machine is very impressive. Sun's biggest selling point with this machine is the 64-bit processing power for under $999 and, without question, you do get a lot of machine for that price. But does a Java developer really need or care about 64-bit processing? I was about to find out now that I was set to begin the trial period.

Now it's fair to say from the outset that as much as I enjoy the Linux/Solaris world for servers, Windows NT is still my preferred desktop choice. So to be forced to use the standard Solaris desktop was indeed a bittersweet pill to swallow, and a pill that was indeed swallowed. I did put up with it, and any criticisms I may pose here are not indicative of the desktop software. I appreciate that you can download and install alternative Windows managers, but I simply didn't have the time to go through this procedure.

However, the pill did indeed have a sugar coating in the form of the preinstalled SunPCi coprocessor card, which at the time of the review was a 600MHz Celeron processor. This was a complete Windows machine within a window and it worked wonderfully. Sun is shipping this dual-computing feature with the Blade to ease the transition for people like myself that are hell-bent on staying with the Windows world, but would like to have the 64-bit processing power and all the other benefits the Blade system can offer.

My only issue with this aspect of the system is that the communication between the virtual Windows machine and the Blade could have been made a lot easier. Ironically, you can copy and paste text between the two with the clipboard, but you can't easily access files. The Blade system's file space is distinct and not directly accessible from the virtual Windows machine. I was informed you can FTP or even set up an SMB/NFS channel between the two, but this isn't set up from the start. You literally have two different machines sharing a keyboard/mouse and monitor, but sharing nothing else. You even need to put in an extra network connection as the ability to share the same network connection doesn't exist.

I believe this to be more of a selling point than an actual practical feature. I couldn't get DVDs or even audio CDs to work, so my initial excitement to get WinAmp up and running was muted very quickly. Now I'm sure much of this is due to configuration and, if this is the case, I hope the final version of Blade is all sorted and ready to run without customers having to dash to the online Web sites for help.

So how did I get on with Java? Well, I copied over one of our internal product sources, totaling around 32,000 lines of code, which I thought would be a good starting point. After firing up Forte and creating the necessary projects, I was happily compiling. No immediate problems. Couldn't really complain.

After a week of this I decided to install a number of application servers on the machine and start testing it from a server's point of view. It's fair to say that Sun was horrified when they heard of this, as the machine wasn't designed as a server but as a desktop. Not too sure where the fire was as it performed admirably, and we put the machine under a lot of stress testing; the combination of Solaris 8 with an UltraSparc 500MHz CPU made light work of our tests.

I took the machine into this world on purpose. Many Linux servers are installed on ex-desktop machines, giving them a whole new lease on life. Seeing the price of the basic Blade system, this brings it into the world of low-end server budgets, and I wanted to see how well it would perform in this world. The results were good and we would happily take a Blade system to run in a server environment.

Should you decide to purchase a Sun Blade, a word of warning. When visiting the main Web site, www.sun.com/desktop/sunblade100/, don't be fooled into thinking you'll be getting a flat-screen panel. It's not part of the standard package, but you have to do a lot of digging on the Web site to discover this. Don't worry too much about the lack of a flat-screen; I wasn't that impressed with it and LCD technology has a long way to go to meet the crispness and sharpness of traditional CRT monitors.

On the whole I liked the machine. I don't think the benefit of 64-bit processing is going to be much of a gain for the average Java developer. I also don't believe you're going to get many developers to give up their Windows machine for a Sun Blade system. If you're a developer who requires a reliable server platform running in the background and doesn't have the budget for a separate machine, the Sun Blade is for you.

That said, it's a fine-looking machine when all's said and done.

Sun Microsystems
901 San Antonio Rd
Palo Alto, California 94303
Web: www.sun.com/desktop/sunblade100/

Sun Blade 100 Workstation:

- 500MHz UltraSPARC-IIe Processor, 256-KB L2 Cache
- 128MB memory (1x128MB DIMM)
- Sun PGX64 On-Board Graphics Accelerator
- 15GB 7200 RPM EIDE disk
- 48x CD-ROM drive
- 1.44MB floppy
- Smart Card Reader
- Solaris 8 Installed (RTU)
- 3 PCI I/O slots
- 2 EIDE disk bays
- 10/100M-bit Ethernet
- 4 USB, 2 IEEE 1394, 1 serial and 1 parallel port

WRITTEN BY AJIT SAGAR

Welcome to J2EE

As Alan Williamson, ***JDJ***'s editor-in-chief, mentioned earlier in this issue, ***JDJ*** 2.0 is a total redesign of ***JDJ***; our intention is to reflect the developments in the Java platform as closely as we can. Some of you may be familiar with my role in ***JDJ*** – I wrote a column, **E-Java**, and was one of the editors. I'm honored to wear the hat of the J2EE editor for ***JDJ*** 2.0. Alan and I have worked together in the past and I'm sure that the synergy between us will take ***JDJ*** 2.0 to new heights – all focused on your needs.

So, without further ado, *Welcome to J2EE*. This section is dedicated to the Java 2 Enterprise Edition and focuses on the application of the Java platform in the development of enterprise solutions. Last year at JavaOne, Sun announced the three Java platform editions. And let me tell you folks, after several years of confusing the community with complicated messages, I think they finally got it right. This past year has been a testimony to the success of this decision. J2EE has matured and come of age.

The past year has also proven to be an eye-opener for those of us developing applications using the J2EE APIs. This stuff actually fits together. For folks developing distributed enterprise applications, it's also easier to focus on the *n*-tier architecture principles evangelized by the J2EE Blueprints to build world-class applications.

The J2EE APIs have also made it easier for application server vendors to publish clearer guidelines for building applications using their products. The last year has seen a shakeout in the app server industry. In 1999, app server vendors were popping up all around the market. With the rude awakening in the B2B market in the latter half of last year and the beginning of this one, several consolidations and mergers have taken place. For example, Bluestone is no longer a stand-alone app server vendor; HP now owns it. The dust is still settling, but the ones who will remain standing are now visible. We have an interview with one of the market leaders, BEA Systems, in this issue. We'll continue to provide you with the latest news from the leading players in the J2EE app server market in subsequent issues.

One thing that has definitely helped the J2EE market is the coherence that's now available in the documentation. Sun has published the J2EE framework in a concise set of blueprints. The types of applications built on J2EE frameworks now include e-marketplace applications such as auctions and storefronts. This is in contrast to the examples on simple airline sites that were available last year. It's now much easier to find information on when to use JSPs instead of Servlets, how CMP compares with BMP, and more.

I'm very excited about this issue. It marks a new era for ***JDJ*** and for our J2EE coverage. In subsequent issues you'll see an assortment of:

- Case studies
- Information from the source (Sun)
- Interviews
- FAQs
- Programmer tips and tricks
- Design guidelines
- Book and product reviews
- A wealth of information that's available only from SYS-CON

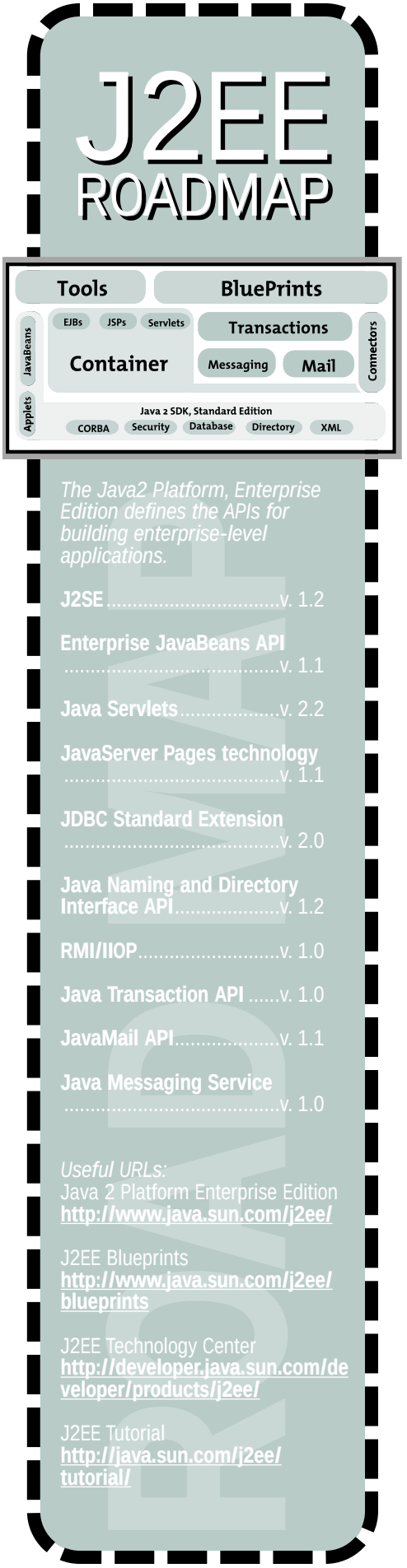
As you know, we've also expanded the venues for sharing information with all of you. Check out our Web site at www.sys-con.com/java and the conferences we're hosting at www.sys-con.com/javaedge.

The only way we can address your specific issues, your requirements, your problems is if we hear back from you regarding what we're doing wrong and what we're doing right. Your input is what differentiates us from the rest. Our sole purpose is to serve your needs. So please write to us and tell us what you think, what you need, and how you'd like us to help you achieve your goals. Feel free to e-mail me at ajit@sys-con.com, or any of the other contributors in this issue, with your feedback.

ajit@sys-con.com

AUTHOR BIO

Ajit Sagar is the J2EE editor of JDJ, and the founding editor and editor-in-chief of XML-Journal. A senior solutions architect with VerticalNet Solutions, based in San Francisco, he's well versed in Java, Web, and XML technologies.



FAQ SPECIAL SECTION: J2EE

What is J2EE and how does it relate to EJBs?

J2EE is a standard architecture to define and support an *n*-tier application programming model for building enterprise applications. It's based on standard design practices and patterns that simplify the application development by decoupling the components between the different tiers of an application. The three main tiers that are defined are the thin client, the business logic layer, and the data source layer. J2EE also encompasses security, performance, and integration issues with external (non-Java) environments.

Enterprise JavaBeans define the server-side component model used by J2EE to implement business logic and are the crux of the J2EE Platform. EJB design defines how to write distributed Java components that run in a J2EE-compliant application server. EJBs simplify development of middleware components by enabling them to be transactional, scalable, and portable. EJB components run only in the context of a J2EE application server, which acts as the "operating system" of the enterprise by handling such mundane tasks as EJB instantiation and destruction, transactions, persistence and recovery, and component deployment.

What are the J2EE Blueprints?

Last year Sun came out with a new set of design guidelines for building enterprise applications using enterprise Java APIs. These are collectively available as a set of documents called the J2EE Blueprints. These documents include architectural design guidelines for developing enterprise applications using the Java 2 Enterprise Edition APIs. The following quote from Sun's Web site describes the rationale behind the J2EE Blueprints:

The J2EE Blueprints provide an integrated set of documentation and examples that describe and illustrate "best practices" for developing and deploying component-based enterprise applications using the J2EE platform. The J2EE Blueprints are intended to help developers of e-commerce applications in the areas of component design and optimization, division of development labor, and allocation of technology resources.

The Blueprints are a great source of information and guidance for developers who are trying to get their arms around J2EE development. They run the gamut of APIs that comprise the J2EE platform. The Blueprints may be viewed in the same light as blueprints that are used to build a house. This set of guidelines outlines the tools at your disposal and the framework that can be used to build applications based on the J2EE APIs. More information about the Blueprints can be obtained from the following sources:

Overview: http://java.sun.com/j2ee/articles/blueprints.html
Download: http://java.sun.com/j2ee/download.html#blueprints
Book: Kassem, N., and the Enterprise Team. (2000). *Designing Enterprise Applications with the Java Platform, Enterprise Edition*, Addison-Wesley.

What is the Pet Store Demo and how can it help me understand J2EE application development?

The Pet Store Demo is an example application built based on the guidelines and design patterns that comprise the J2EE Blueprints. In other words, the Pet Store Application is akin to a model house built using the J2EE Blueprints.

The Pet Store is a complete end-to-end e-commerce application built using J2EE. However, it is a sample application. The demo acts as a great tutorial for understanding how to build Web-based e-commerce applications using J2EE. However, to build a real-world application, you will have to deal with issues including security, scalability, and performance, using the demo as a base. Some of the Java application server vendors have implemented the Pet Store Demo as a sample application built using their respective application server features.

More information on the Pet Store can be obtained from the following sources:

Architecture Overview: http://java.sun.com/j2ee/blueprints/jps11/archoverview.html
Book: Kassem, N., and the Enterprise Team. (2000). *Designing Enterprise Applications with the Java Platform, Enterprise Edition*, Addison-Wesley.
Application Servers: BEA Systems offers a tutorial using the Pet Store Demo as a base. The demo is documented in their standard documentation sets.

Interview with Larry Freeman

Manager, J2EE Blueprints Team

INTERVIEWED BY AJIT SAGAR

As readers of ***JDJ*** know, J2EE is a standard platform for developing enterprise applications using reusable components, standard APIs, and popular software design patterns. The J2EE Blueprints is a Sun initiative that's meant to aid developers trying to get their arms around the wide gamut of J2EE technologies and APIs, and the Pet Store Demo is a part of this initiative. They provide a combination of J2EE design guidelines and a sample application for building distributed applications on the Java platform. This month ***JDJ*** brings you an exclusive interview with Lawrence Freeman, the manager of the J2EE Blueprints team.

JDJ: **The last year has seen some major developments in the J2EE Platform. Were there any targets Sun wanted to achieve in this time?**
Freeman: There are four deliverables for J2EE: the specification, the reference implementation (RI), the Compatibility Test Suite (CTS), and the reference application, that is, the J2EE Blueprints.

JDJ: **What is the Pet Store Demo?**
Freeman: The Pet Store Demo is the reference application for J2EE that showcases the main features of the J2EE platform. It's part of the Blueprints program that seeks to provide best practice guidelines for J2EE application development. J2EE Blueprints includes a Web site, a best-selling book, design patterns, and reference applications. In the future, J2EE Blueprints will include other demo applications besides the Pet Store. The Pet Store is currently being used in the Deployathon, an event in which a J2EE application is deployed and run on multiple application servers to demonstrate J2EE portability.

JDJ: **Could you briefly describe what role you've had in the design and "disbursement" of the Pet Store Demo application?**
Freeman: I'm the manager of the J2EE Blueprints team, and my team designed and wrote the code for the Java Pet Store. As J2EE's Application Programming Model group, we're responsible for making best-practice recommendations to application developers. The Java Pet Store was designed to illustrate these guidelines. The J2EE Blueprints team will be speaking at this year's JavaOne.

JDJ: **The Pet Store demo is an unusual application. How did it emerge?**
Freeman: The application began as part of an effort to write a book on the J2EE application programming model. A sample application was needed to go with the book.

The original application was a wine store; however, the images that represented the different wine bottles all looked the same. Two members of our team who are pet lovers thought a pet store with its variety of animals would make a more interesting demo.

The important idea is that an online store is one of many application scenarios that can be built using the design guidelines presented by the Blueprints team.

JDJ: **Could you broadly describe how the Pet Store application can be used to build and deploy real-world applications?**
Freeman: I think the key here is the design guidelines and patterns that are presented on the Blueprint Web site, http://java.sun.com/j2ee/blueprints. The goal is to help developers and architects make design decisions when writing a J2EE application.

Implicit in the Java Pet Store is a set of components that can be reused in other contexts. For example, the Java Pet Store includes a shopping cart, a mailer, and inventory components. It's our intention that this reference application can be used as a framework for building real-world applications. In addition, recommendations about typical design decisions need to be made, such as JSP versus servlets, stateful versus stateless session beans, and container- versus bean-managed persistence.

JDJ: **The Pet Store serves as a good reference implementation. However, in the real world do you expect the application writers to implement all the pieces of the application or do you expect part of the framework to be built by third-party vendors such as application server providers?**
Freeman: It's our belief that these decisions are best left to the market. I think there are many fascinating frameworks out there. For example, in the recently released J2EE Tutorial (http://java.sun.com/j2ee/tutorial), there are some examples of the Struts tag library (http://jakarta.apache.org/struts) that can help developers speed up the implementation of their applications. Other tag libraries can be found at http://java.sun.com/products/jsp/taglibraries.html. It's our hope that the Pet Store and other Blueprints demo applications will aid the developer and architect in building J2EE applications that take advantage of key features.

JDJ: **What is Sun's relationship with J2EE application server vendors with respect to the reference implementation?**
Freeman: The reference implementation is not an industrial-strength J2EE implementation. It's meant as a way for the developer community to try out J2EE and write applications on it before purchasing a professional J2EE platform. For example, there's an early access reference implementation currently available that enables developers to try out J2EE 1.3 (http://developer.java.sun.com/developer/earlyaccess/j2ee). The reference implementation is also used by J2EE application server vendors as a standard for comparing their own implementations.

JDJ: **How many application server vendors have implemented the Pet Store as a reference application that's distributed as part of their product?**
Freeman: I assume you're referring to the J2EE Deployathon. Last year at the Deployathon an earlier version of the Pet Store was run on numerous J2EE application servers to demonstrate J2EE portability – it was tested on eight different platforms. Detailed information about last year's Deployathon can be found at http://developer.java.sun.com/developer/technicalArticles/J2EE/deployathon2.

In this year's Deployathon we expect to have a larger number of participants using Java Pet Store 1.1.2. We'll have full details at JavaOne.

At present, I'm not sure about the exact number of application server vendors who are distributing the Pet Store as part of their product, however, it's available for download at our Web site. For the latest information on J2EE-

> The J2EE reference implementation is an excellent way to try out J2EE. **Best of all, it's free.**

compatible application servers, check out http://java.sun.com/j2ee/compatibility.html.

***JDJ*: What exactly is the Deployathon and how is it used?**
Freeman: The Deployathon is an exercise run by Sun to demonstrate J2EE's "write once, run anywhere" message. It takes the Blueprints reference application and deploys and runs it on different J2EE app servers. Last year's Deployathon is available online at http://developer.java.sun.com/developer/technicalArticles/J2EE/deployathon2.

***JDJ*: Does Sun provide standard guidelines for vendors to implement the Pet Store?**
Freeman: I'm not sure what you mean by "implement." Java Pet Store is designed to be deployable and runnable on all J2EE-compatible servers. We provide both the source code and the binary form of the Java Pet Store on our Web site. Any developer on any J2EE-compatible application server can run the reference application on their platform.

***JDJ*: What types of problems have vendors faced getting their application server products to comply with the J2EE Blueprints that are used to implement the reference application?**
Freeman: J2EE Blueprints is a set of design guidelines used by application developers. It's routine that after an application server has passed the CTS, the next step is to verify that the Java Pet Store can run on the platform. This is a de facto test; the CTS is the official test. However, none of this is required by the J2EE Blueprints.

***JDJ*: Have you received any feedback from the vendors?**
Freeman: We've received a large amount from vendors and developers. As the J2EE best practice group, we're constantly thinking and rethinking the best way to implement the different J2EE technologies in applications. For example, we've updated our book with more recent design recommendations based on this feedback, which can be found at http://java.sun.com/j2ee/blueprints/qanda.html.

***JDJ*: How useful do you think such a reference application is for the Java community? Does it benefit only the J2EE platform developers or can it be used by all factions of the Java community?**
Freeman: My team is focused on the J2EE so their recommendations focus on it too.

***JDJ*: Does the Pet Store Demo contribute to any type of certification or measuring stick for Sun to evaluate application server vendors?**
Freeman: It's not a measuring stick for Sun. I'd say it's more of a tool used by the application server vendors themselves. As mentioned earlier, it's a useful test after a vendor has passed the CTS. It's a standard way for them to try their J2EE platform out. Sun's official test is the CTS.

***JDJ*: Of the vendors who have leveraged the Pet Store to provide tutorials for their products, which implementations do you think offer the best examples?**
Freeman: This is not the intention of the Pet Store. The goal of the Blueprints program is to help developers take advantage of the key features of J2EE. Our target audience is developers.

***JDJ*: Are there any other reference applications in the offing? Do you think the Pet Store covers it all?**
Freeman: There are other reference applications currently being planned. I'm very excited about what's coming up. I'd encourage developers to continually check http://java.sun.com and http://java.sun.com/j2ee to find out the latest. In addition, developers can sign up for our mailing list at http://archives.java.sun.com/archives/j2ee-interest for the latest information.

No, I don't think the Pet Store covers it all. That's not its goal. Its goal is to cover all the major features of the latest release of J2EE. When J2EE 1.3 is released, a reference application will also be released that showcases its major features.

***JDJ*: How can our readers familiarize themselves with the Pet Store? What if they don't have access to an industry-strength application server?**
Freeman: I'm glad you asked this question. The J2EE reference implementation is an excellent way to try out J2EE. Best of all, it's free. It's one of the many platforms that make up the Deployathon.

Another way to become more familiar with the Java Pet Store is to take advantage of the Blueprints book available at http://java.sun.com/j2ee/blueprints/apmtoc.html.

***JDJ*: What other resources are available for learning J2EE application development?**
Freeman: At present J2EE is putting together a Developer's Corner that should be available before JavaOne; check for this at http://java.sun.com/j2ee. As I mentioned before, there's a new J2EE tutorial available at http://java.sun.com/j2ee/tutorial. Another useful resource is the document bundle that comes with the reference implementation. Included in this bundle is the J2EE Developer's Guide, which helps developers take advantage of the Deploytool, a GUI-based tool that comes with the reference implementation. The Java Developer Connection has resources for learning more about J2EE, for example, http://developer.java.sun.com/developers/technicalArticles/J2EE. For developers looking for an IDE, there's an early-access release of the J2EE IDE Toolkit at http://developer.java.sun.com/developer/earlyAccess/j2eeide-toolkit.

***JDJ*: Are there other "Blueprints" under development for the Java platform? Does Sun plan to provide other implementations of them?**
Freeman: Yes, there are other other implementations. Hopefully, we'll be able to make some announcements at JavaOne. In the meantime, keep checking our Web site at http://java.sun.com/j2ee/blueprints.

WebLogic Server 6.0

by BEA

REVIEWED BY JIM MILBERY

AUTHOR BIO

Jim Milbery is a software consultant based in Easton, Pennsylvania, with Kuromaku Partners LLC. He has over 17 years of experience in application development and relational databases. He is the applications editor of Wireless Business & Technology, *the product review editor of* Java Developer's Journal *and the author of* Making the Technical Sale.

apps@sys-con.com

Test Environment:
Dell 4300 PowerEdge 2CPUs (397MHz), Windows NT Server 4.0 SP5 1 GB RAM

BEA Systems, Inc.
2315 North First Street
San Jose, CA 95131
800 817-4BEA
www.beasys.com

Bill Coleman, Edward Scott, and Alfred Chuang must be looking at their September 1998 acquisition of WebLogic as the best money they ever spent. WebLogic's Tengah product was a little-known, Java-based application server when BEA made the decision to buy their way into the growing market for Java application servers way back when. Since those early days the J2EE specification has matured and BEA has made great strides with the WebLogic product line. Their most recent effort is WebLogic Server 6.0 – a product that was touted with much fanfare at BEA's eWorld conference in Dallas, Texas.

Fresh from my indoctrination at eWorld, ***JDJ*** turned me loose on a copy of WebLogic Server 6.0 to give it a test drive.

In my opinion, the market is starting to see a division between the enterprise-class application servers (IBM, Oracle, iPlanet, and BEA) and other Java-based application servers. The enterprise J2EE servers are stocked chock-full with lots of bells and whistles, while the remaining products tend to offer "best-of-breed" solutions for specific application needs – such as JMS, servlets and JavaServer Pages. BEA's WebLogic 6.0 is clearly an enterprise-class product, and it continues to take the lead in all things related to Java as far as application servers are concerned. I've tested numerous versions of WebLogic in the past, and I've always found the product to be easy to install and configure. Each version of WLS offers new features, which in turn makes the software more complicated to install, manage, and maintain. This is not to say that BEA has not made incremental improvements in the management tools for WLS. Clearly, the new management console, shown in Figure 1, is well organized and robust.

In fact, all of the panels in the management interface are hyperlinked to the online documentation, so it's easy to work with the administration panels. The issue is just how complex and sophisticated these enterprise-class application servers are fast becoming. BEA is supporting a wealth of functionality within the WLS 6.0 product, and there's just no getting around the fact that it's a lot of product to get your arms around. WLS 6.0 is not a product that you can expect to master overnight. BEA continues to forge ahead in the J2EE realm and WLS 6.0 supports a wealth of new and enhanced Java features such as EJB 2.0, JMS, XML, and native JDBC drivers for most of the popular RDBMS engines. However, the new release also offers improved functionality in several other critical areas of the product. We've already mentioned the updated Web administration interface (based on Sun's Java Management Extension: JMX), which is substantially better than previous releases. BEA now provides support for HTTP 1.1 directly within WebLogic, so there's no need to install or configure a separate Web server. While this is not a critical feature for production sites (that may have already standardized on Apache/iPlanet or Microsoft IIS), it makes life a lot easier for developers to work with WebLogic. This release also includes a number of security enhancements such as ACL improvements, audit trails, and JAAS login support.

During the user conference I got the impression that WLS (rather than WLE) is quickly becoming the key product for BEA. However, underneath the covers BEA is clearly drawing from their Tuxedo expertise to flesh out the WLS platform. Updated transaction-processing features, such as distributed transaction management, are just one example of this effort. BEA was an early proponent of advanced scalability features for the application server platform (another result of their Tuxedo heritage). WLS 6.0 extends its support for servlet replication to include replication for session EJBs. This new release can even perform in-memory replication of stateful session EJBs, which provide failover for business logic. In the long haul, BEA is staking out the high ground as regards scalability and fault tolerance. The customers that I spoke with have not necessarily taken advantage of these high-end features in previous releases. (In fact, I have sat through several of Dean Jacob's highly regarded user group sessions – and Dean's advice is always the same. If you want scalability, design your applications to be stateless.) With the release of WLS 6.0, the scalability features appear to be more fully baked in, and I would venture a guess that developers will begin to leverage these features more fully in their applications. Servlet failover and EJBs tend to be the two big features that I hear developers talk about when they mention WLS 6.0.

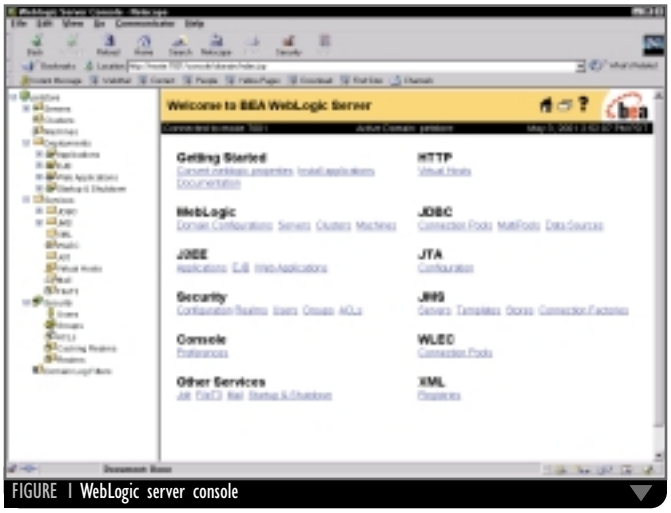
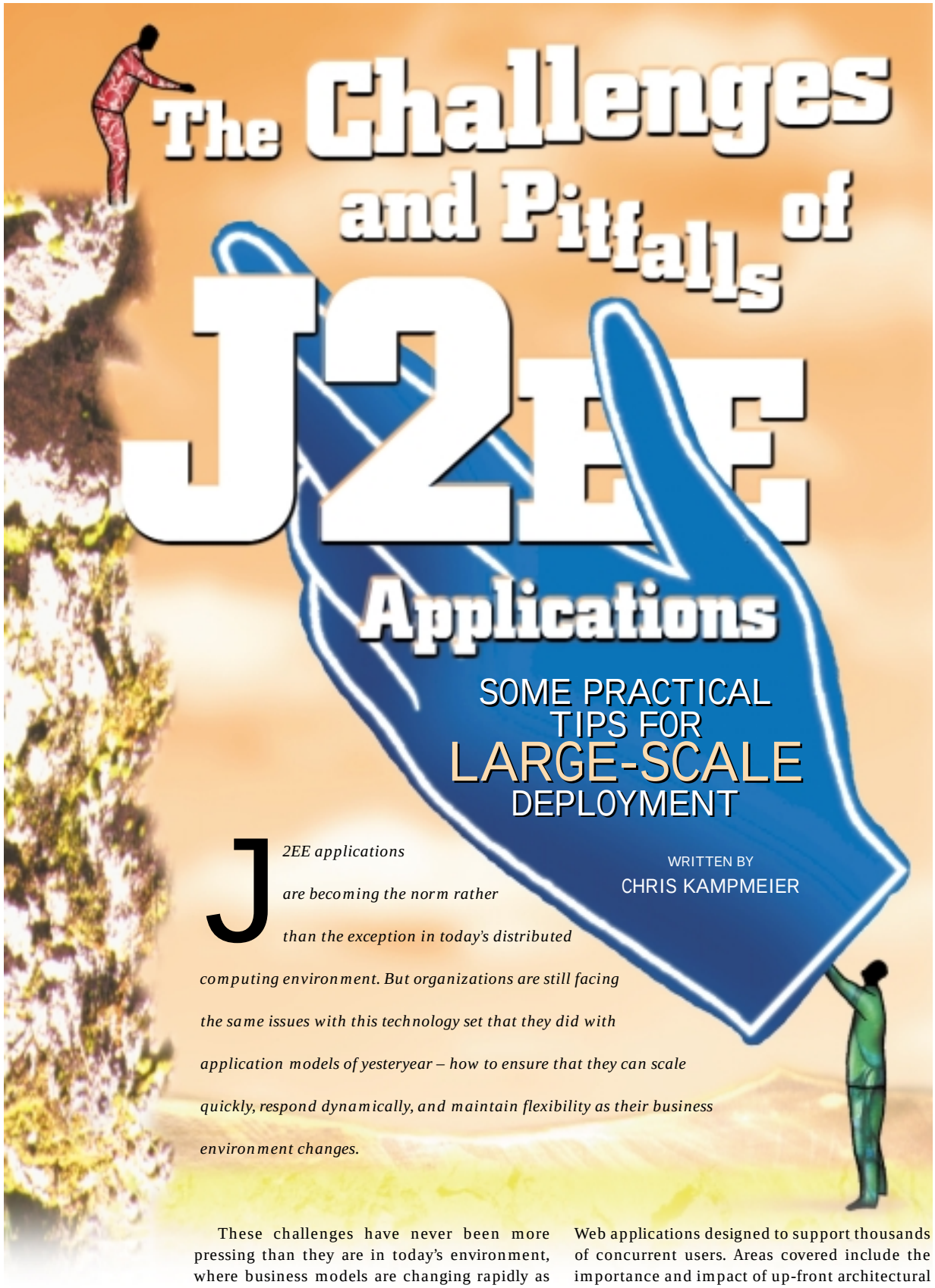


FIGURE 1 WebLogic server console

Summary

BEA's WebLogic Server 6.0 continues to break new ground in the J2EE application server market. If you're looking for a high-end application server with extensive transaction processing capabilities, WLS 6.0 is worth a long look. You can download a free 30-day trial from BEA's Web site.



The Challenges and Pitfalls of J2EE Applications

SOME PRACTICAL TIPS FOR LARGE-SCALE DEPLOYMENT

WRITTEN BY
CHRIS KAMPMEIER

J2EE applications are becoming the norm rather than the exception in today's distributed computing environment. But organizations are still facing the same issues with this technology set that they did with application models of yesteryear – how to ensure that they can scale quickly, respond dynamically, and maintain flexibility as their business environment changes.

These challenges have never been more pressing than they are in today's environment, where business models are changing rapidly as organizations cope with the realities of a cyclical economy.

This article focuses on some of the application characteristics of large-scale J2EE applications and is prescriptive in terms of what we have seen work and not work with large-scale Web applications. In particular, this article considers Web applications designed to support thousands of concurrent users. Areas covered include the importance and impact of up-front architectural decisions, development steps to help ensure smooth deployments, performance tuning, and production deployment planning and design.

Considering Architectural Approaches

Because of the popularity of both Java and J2EE, organizations have an unprecedented vari-

ety of portable software to draw on in the formation of their architectural standards. Since J2EE currently focuses more on application portability and less on the underlying implementation and operational characteristics of an application server, it's important for organizations to look beyond the J2EE programming model and concentrate on how key decisions will impact the operational characteristics of the deployed systems. Apart from decisions related to the underlying hardware and operating systems, the following architectural considerations will weigh most heavily on the performance of the operational environment.

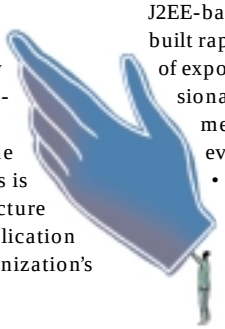
- ***Select application server:*** To an extent not seen with other Web and enterprise component platforms, the J2EE platform enables an organization to base the bulk of its development investment on the portable base platform and layered frameworks rather than on vendor-specific APIs and features. The adoption of J2EE has allowed organizations to worry less about comparing application development environments, but they still need to perform a comprehensive evaluation of the available J2EE-compliant application server products to ensure that the products meet the organization's operational requirements. Switching application servers midway through development of a J2EE-based application effort will moderately impact the delivery schedule. However, given the investment necessary to train operational teams on the product-specific processes and features, switching application server products near or during the deployment phase will result in significant additional costs and extended deployment delays.
- ***Examine Web presentation layer frameworks:*** Almost any Web application will benefit greatly from the use of a prebuilt Web presentation layer framework. For example, the Apache Jakarta project's Cocoon and Struts Web presentation layer frameworks have become very popular with many J2EE developers. Although some teams will be hesitant to build the Web presentation layer on a third-party framework, most organizations end up developing a generic infrastructure that sits between the base J2EE Web container services and business presentation logic – whether or not they call it a framework. In addition to the time savings they generate, such frameworks are portable across almost any J2EE Web container implementation. Although the popular frameworks evolved with JSPs and servlets prior to the formal J2EE specification, the relative newness of the frameworks warrants caution when considering them for production use. It's advisable to thoroughly test any adopted framework in a production-like environment to ensure that stability and performance requirements are met.
- ***Survey prebuilt components:*** As the architecture team identifies enterprise components that will be useful to many of their organization's business applications, the architects should first review the prebuilt Java and EJB components. This is true whether they're available through their application server vendor or through the several component marketplaces on the Web. As in the case of prebuilt frameworks, third-party components, even if customized, gain greatly from speed development efforts. The Flashline.com site references a wide variety of both EJB and Java components. As part of any component evaluation effort, each component should be thoroughly reviewed and tested for its performance and security characteristics.
- ***Make hard decisions on RDBMS integration:*** One of the most contentious aspects of J2EE and EJBs is RDBMS access. It's worthwhile for an architecture team to invest a great deal of time with key application design team members to determine the organization's requirements with respect to RDBMS integration. This should be done even before testing the various approaches. Since approaches can run the gamut from lightweight, local-access DAO-style classes to heavier-weight, remotable CMP-based entity beans, there are huge design and implementation ramifications to adopting one approach over another. For example, using entity beans enables developers to rely on the underlying EJB container to manage complex aspects such as transactions, security, and remote communication. However, the container's role comes with the price of additional processing overhead. Many organizations fail to realize the downstream impact of their RDBMS integration approach on system performance until well into the deployment phase of an application.
- ***Decide on enterprise connectivity solutions:*** From the standpoints of both development efficiency and runtime performance, it's generally preferable to employ prebuilt back-end connectors to shield developers of the Web presentation layer and new EJB-based business components from the detailed interface semantics of existing systems. Much like the RDBMS integration, back-end connectivity to enterprise systems can be a major source of headaches during deployment if the proper evaluation and testing of solutions is not performed in advance of development. Although the emerging Java Connector Architecture (JCA) is gaining adoption as a standards-based approach to enterprise connectors, many of the higher-end application servers already support some degree of prebuilt, albeit proprietary, connectors. A few of those vendors offer SDKs to enable organizations to quickly build their own connectors for back-end systems not addressed by the prebuilt connectors. Compared to relying on individual developers or development projects to implement efficient and secure access to back-end systems from their J2EE applications, the connector approach enables an organization to ensure consistent performance and security characteristics across applications.

For J2EE-based applications, these are several of the most prominent architectural considerations. In each case, upfront investigation and testing will help ensure a solid base for development and deployment.

Designing and Developing Production-Ready Applications

The source of perhaps the largest impact on the viability of a J2EE application in production is the initial detailed design of the application. Even though many development teams are new to J2EE features, we often encounter cases in which the team performs very little, if any, performance testing of individual application components, let alone the complete application, prior to the production deployment phase. In truth, there's nothing magic about J2EE that would permit a development team to skip important steps in the basic development process. Key steps worth highlighting are:

- ***Engage experts:*** For your first J2EE development effort, engage experts that have already implemented production J2EE-based systems. Since momentum behind J2EE has built rapidly and the standard has had approximately a year of exposure, there are now many more experienced professionals available to lead new teams through development and deployment of applications than there were even nine months ago.
- ***Leverage proven design patterns:*** Become familiar with the popular design patterns that apply to your application problems. Proven patterns have typically evolved to the point of becoming the most



efficient yet maintainable means of addressing a particular problem. A set of patterns applied to J2EE-based systems was recently published on java.sun.com at http://developer.java.sun.com/developer/technicalArticles/J2EE/patterns/.

- ***Consider impact of session size on performance:*** If you're considering the use of session-replication features provided by the application server, consult the session-size guidelines provided by the application server vendor. These guidelines are important when determining how much information to store in the HttpSession of the Web application and/or instance variables of stateful session beans. Although management of large amounts of session data may yield acceptable performance when an application is deployed to a single application server instance, the performance characteristics may vary dramatically as session replication is configured at deployment time. This is often done in an effort to ensure availability of session data across system outages.
- ***Perform code walkthroughs:*** It's critical that development teams review source code to ensure that resources are properly managed by the application. The J2EE environment provides many opportunities for sloppy housekeeping. Code reviews are an early and, therefore, relatively low-cost means of avoiding the following common mistakes:
  –Failure to close JDBC result sets, statements, and connections
  –Failure to remove unused stateful session beans
  –Failure to invalidate HttpSession

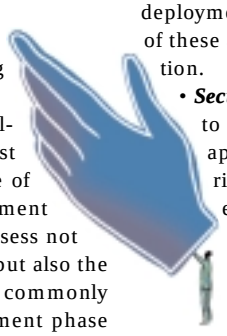
  Although the JVM will handle the garbage collection of unused object instances, the application server product and JDBC drivers implement their own strategies for cleaning up unused resources. To minimize the overhead imposed by such housekeeping tasks, it's important for the application to clean up after itself. If code reviews are not performed, then the next opportunity for catching resource issues is during either code profiling or initial tuning efforts.
- ***Performance test various options:*** In your development environment, establish small-scale test harnesses to evaluate the relative performance of different configurations. For example, experiment with both Type 2 and Type 4 JDBC drivers to assess not only their compatibility with your application, but also the performance of each driver type. Use any of the commonly available load-generation tools in the development phase to simulate moderate loads against a portion of your application. Use the application server monitoring tools to gain insight into resource utilization prior to formal system and integration testing.
- ***Perform code analysis and profiling:*** One of the biggest issues arising during the deployment of new J2EE-based applications is the lack of expected performance. Often, such performance problems can be readily identified in advance through the use of mature code profiling tools such as JProbe and OptimizeIt!. Although these products predate J2EE, they are able to hook into the leading application servers to provide a comprehensive view of code efficiency.

Understanding Operational Requirements

As the application development activity ramps up and you have begun to tune simple deployments of the application, the implementation team can establish the general layout of the operational environment based on the requirements of the application server product and the business application. This general layout will show the relative position of the system's external interfaces. Before the general layout can be transformed into a more concrete design, implementers need to factor in the following operational requirements of the new system:

- Security
- Availability
- Performance

Mature organizations are well schooled in these facets of an operational environment. One of the key differences between J2EE-based application servers and older middleware is that high-end application servers provide many deployment configuration options that drastically impact all of these areas without affecting the application implementation.

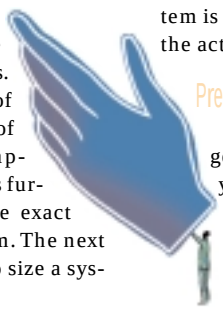


- ***Security:*** Is there a requirement to encrypt the browser to Web server communications for all or part of the application? Will the Web-server tier exist in a demilitarized zone (DMZ) separate from the application-server tier and back-end enterprise systems? Is encryption required between the Web servers and application servers? If so, is the encryption necessary for all interactions with the application? These are questions you have to consider.

- ***Availability:*** Availability issues are also critical. What are the availability requirements of the application? Is the loss of service acceptable when a machine becomes unavailable? Can the loss of a user's session information be tolerated? What are the possible weak links with respect to the manner in which the application interacts with other aspects of the environment? Can these weak links be reasonably enhanced or are they constants? Many shops view a CPU busy rate of 80% as a high-water mark. What is your shop's standard?
- ***Performance:*** Performance requirements pose challenges as well and must be addressed. What, for example, are the required response times experienced by the end users for various interactions with the application? What are the perceived steady state and peak user loads? What is the average and peak amount of data transferred per Web request? What is the expected growth in user load over the next 12 months?

For peak user loads, you must focus on the number of concurrent sessions being managed by the application server. We often find that organizations view peak user load as the number of possible users rather than the average number of concurrent users. Given this more realistic view of user loads, you'll often find the number of peak users drops dramatically on paper from hundreds of thousands or even millions to tens of thousands of concurrent users.

Defining these operational requirements will help move you to the next stage in understanding the deployment environment. For the purposes of this article, let's assume that the operational security and availability requirements are such that multiple Web and app server instances, separated by a set of firewalls, will form the basis of the environment. In other words, you'll need separate tiers of machines to support the division between the DMZ and the secure, back-end business systems. You'll also need to plan on multiple instances of machines in each tier to enhance the availability of the application. Proceeding from these assumptions, the layout of the operational environment is further refined, though it does not yet address the exact number or size of machines required by the system. The next step is to develop a basic understanding of how to size a system.



Understanding Factors Affecting Sizing

The main factors affecting sizing are:

- ***User load:*** The larger the load that's applied to a system, the greater the amount of hardware needed to satisfy that load.
- ***Application design/implementation:*** An application that performs very little work will be able to handle many users for a given amount of hardware. In a relative sense, this kind of application often scales poorly as it spends a large percentage of its time waiting for shared resources (network, database, other enterprise systems, etc.). Conversely, applications that perform a great number of computations tend to require much more hardware per user, though these applications typically scale much better than those performing a small number of computations.
- ***Hardware platform:*** Raw processor performance is critical for reducing the amount of hardware needed. Generally, Web applications do not include floating-point-intensive computation, which means integer performance is usually the most important factor. Even with high-speed processors, a server can scale poorly if shared resources cause significant contention. Usually, cache design and memory bandwidth play a big role in determining how much extra performance is achieved, as processors are added to a server.
- ***Safety margins:*** Additional capacity is normally designed into a solution. One reason for this is that user loads tend to increase over time. Another reason is that most businesses expect to keep their Web-based services available during planned and unplanned outages. Proper sizing and capacity planning is typically able to compensate both for the increase of loads as businesses grow and for partial system outages. Planned outages of a portion of the system may also be required as this permits an application system to be upgraded to a new version. To do this, a portion of the system is taken offline, while user requests are still handled on the active portion of the system.

Predicting Performance

Given the factors affecting the sizing process and the general layout of the operational environment, how do you predict either the capacity of a given combination of hardware or the minimal hardware required to sustain a specified capacity? The best way to answer these questions is to take the data gathered above and plug it into the sizing calculator offered by your

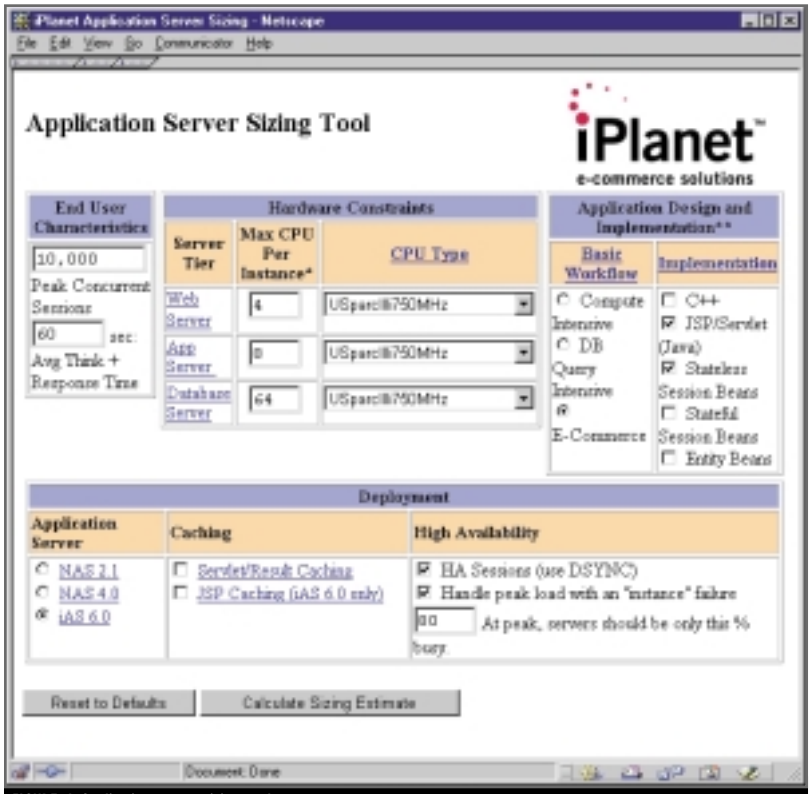


FIGURE 1 Application server sizing tool

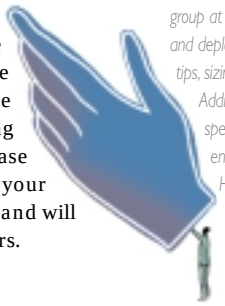
application server vendor. For example, iPlanet provides customers with two calculators to help size applications deployed to the iPlanet Application Server. The first calculator computes the size of a system (i.e., the number of CPUs and the number of machines in each tier) based on the factors described above. The second calculator computes the maximum capacity of a given hardware configuration (see Figure 1).

iPlanet built these calculators based on a combination of tests, including those for popular application workflows, and drew upon publicly available benchmark results for RDBMSs and processors. Both calculators assume a fully tuned system.

If your application server vendor does not make a calculator available, or the application workflows do not match those of your application, then you'll need to develop your own understanding of sizing based on the following steps:

1. ***Determine performance on a single CPU:*** You must determine the largest load that can be sustained with a known amount of processing power. You can obtain that figure by measuring the performance of the application on a uniprocessor machine. You can either leverage the performance numbers of an existing application with similar processing characteristics or, ideally, use the results of basic performance testing done during development. Based on their experience with large-scale J2EE implementations and internal validation of their products, vendors of high-end application servers can usually provide base performance numbers for Web applications implementing a basic workflow.

   While determining performance on a single CPU, it's imperative that you begin to tune the basic environment. As with any performance test, you must ensure that none of the outlying systems (driver machines, Web servers, database machines, etc.) throttle the test. Otherwise, your performance numbers may be artificially low and will adversely impact the follow-on sizing numbers.



2. ***Determine vertical scalability:*** You need to know how much additional performance is gained when you add processors. That is, you're indirectly measuring the amount of shared-resource contention that occurs on the server for this workflow. You either obtain this information based on additional load testing of the application on a multiprocessor system or leverage existing information from a similar application that has already been load tested. Running a series of performance tests on one to four CPUs generally provides a decent sense of the vertical scalability characteristics of the system. On Solaris, for example, it's easy to disable/enable processors. Based on your sizing estimates, it's important to exercise the application under load on systems of the target configuration.

   While determining the vertical scalability, ensure that availability requirements are factored into the configuration. For example, to guarantee that the failure of a single JVM does not result in a loss of all sessions, perform the vertical scalability tests with at least two JVMs and configure session replication between the JVMs.

3. ***Determine horizontal scalability:*** You need to know how much additional performance is gained when you add servers. Again, benchmarking a cluster of application server systems is required if information on a similar application is not already available. Ensure that you take into consideration high-availability requirements and the attendant session-replication configuration as you lay out your horizontal scalability test environment. In this case, session replication occurs across application server instances deployed to multiple machines, in addition to session replication across JVMs within each application server instance.

Running this suite of tests will provide you with a solid understanding of the performance of the application server. Using this information, you can develop your own custom-sizing equations.

Summary

The increased programming flexibility and the great number of deployment time configuration options offered by J2EE and advanced application servers can cut the time it takes to develop and deploy highly available business services. Before these benefits can be realized, however, mature development and deployment practices must be established. In short, J2EE provides a powerful development paradigm, but one that requires careful operational planning to ensure success.

Author Bio

Chris Kampmeier is group manager for technical evangelism within the application services group at iPlanet E-Commerce Solutions. He leads the team responsible for creating development and deployment content for the J2EE developer community including sample applications, coding tips, sizing and tuning guides as well as documentation for the application server family. Additionally, Chris is heavily involved in defining future product strategy. Prior to iPlanet, he spent more than two years at Sun Microsystems supporting field systems engineers, and seven years developing EFT switching systems at MasterCard International. He holds a BS in computer science from Northern Illinois University.

ckamps@iplanet.com

GROWING JSP AND SERVLET SITES INTO EJB-BASED SERVICES

WRITTEN BY
PATRICK SEAN NEVILLE

Java Servlets and JavaServer Pages (JSP) threaten to collapse out there in the mud houses of server land. Developers succumb to the temptation to muddle their pages with complex business logic, and to fill behemoth proprietary libraries with data and subsystem access routines. We find stronger solutions in EJB-based architectures than in fattened JSPs, but molding existing sites into EJB-centric designs strikes most of us as a daunting and expensive task.

THE POSSIBILITIES ARE POWERFUL WHEN THE DESIGN AND IMPLEMENTATION ARE **ATOMICALLY ELEGANT**

However, there are clear guidelines for evolving the JSP and custom code approach into a more flexible EJB-based architecture. Once convinced that the endeavor is both worthwhile and feasible, developers can take steps to pare down servlets and JSPs to simple MVC-compliant presentation layer elements that elegantly access hard-core logic in the form of decoupled and refactored EJBs. This is the practice of growing a JSP site into a full-blown J2EE application.

JSP Anti-Patterns

In designing software, most of us will agree that mixing presentation logic and business logic is a naughty thing to do. In practice, however, JSPs do not naturally lend themselves to a decoupling of these layers, often leading us to compromise our design aesthetics in favor of implementation convenience. After all, we're not HTML designers. We know Java, and we don't mind plopping Java routines right in the middle of our pages. It's quick and dirty. Eventually, as the business logic in the pages grows more complex and touches more external subsystems (databases, mail servers, custom proprietary servers), our cool apps coalesce into great balls of mud. Figure 1 illustrates such a scenario, in which the client event handlers perform business logic that includes enterprise-level resource usage.

It's not that a pattern has become obscured in such cases; it's that an anti-pattern has emerged. Anti-patterns begin as compelling solutions that result in the eventual amplification of the problems they address. In particular, many JSP and servlet sites follow the BLOB and Stovepipe anti-patterns, in which originally compelling code spirals out of control and sprouts numerous tightly coupled subsystems. Designers,

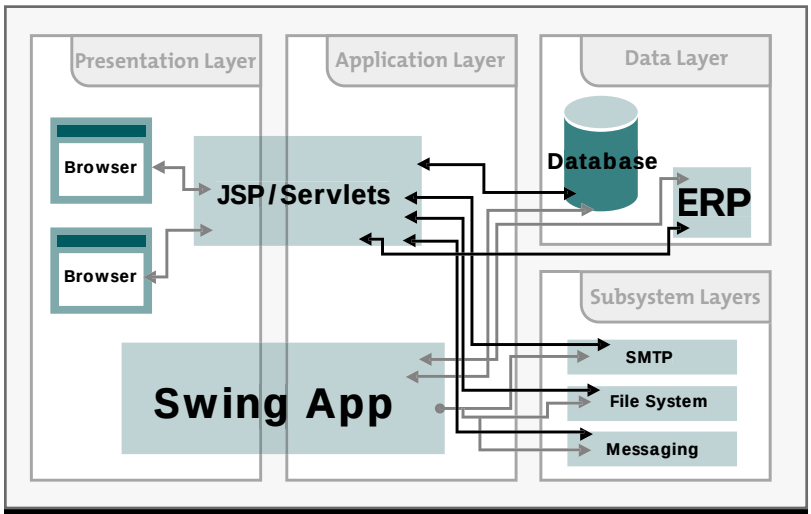


FIGURE 1 A typical first-generation JSP/servlet Web application

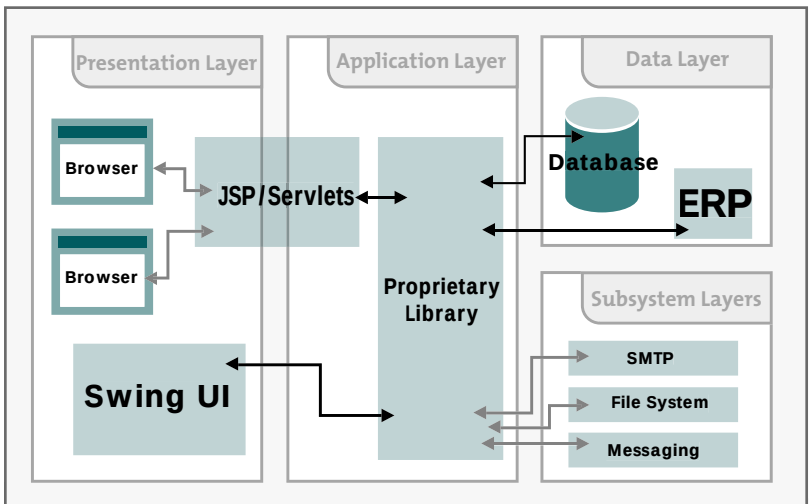


FIGURE 2 The enterprise logic is refactored into proprietary libraries

developers, and engineers step on one another's toes to turn simple systems into sophisticated ones. The Web apps developed in this manner are akin to a house in which cleaning the windows inexplicably causes the roof to leak. The situation grows even uglier when new teams take charge of maintaining the code. Justifying the code BLOB under the "Controller" MVC label doesn't resolve the practical problem, even if it sounds promising in high-level architectural discussions.

Lengthy procedural code is more difficult to test than object-oriented code, and dropping business logic into JSPs and servlets eventually results in just this sort of procedural code. Unit and regression testing is a snap when a specific function is isolated in layered component architectures, but QA balks at tightly coupled systems that do not readily expose functionality to clean test cases.

To resolve these problems, many developers take the pre-EJB route of moving complex business logic out of a few JSPs and into chained servlets and separately included JSPs, and eventually into proprietary libraries and utilities that their pages invoke. Proprietary libraries typically consist of scores of classes linked in complex inheritance schemes that grow increasingly convoluted over the library's lifetime. These libraries in turn may grow to call upon bandied-together O/R tools, or transaction engines. Figure 2 illustrates this scheme, in which proprietary libraries assume the burden of security, transactability, persistence, and resource connectivity in addition to the application logic. This approach not only exacerbates the issue by spreading the problems out among a larger code base; it also implies instant legacy code – costly and confusing to maintain, not portable, proprietary and interoperable with nothing beyond its own boundaries. In anti-pattern terms, the Stovepipe systems and BLOBs are eventually covered by unmaintainable lava flow.

Tag libraries, or taglibs, are a terrific solution for many problems caused by typical JSP usage, but when taglibs contain hard-core business logic – requiring transactions and concurrent processing, for instance – they suffer the same problems associated with all proprietary code libraries. Taglibs are excellent presentation layer elements that cleanly broker communications to portable remote business logic components, but they should not contain business logic and resource connector code. Your tag-loving Web designers will be much happier with a handful of powerful tags that can access business components rather than hundreds of overly specialized tags that perform the business logic themselves.

When masterfully employed, servlets and JSPs conform to event-based rather than component-based patterns. Servlets are event processors; in comes a request, out goes a response. They thrive in pipe-and-filter designs, in which the request and response types and the sequence of pipes through which they flow define the system functionality. Therefore they thrive as the transformers of client/server communications.

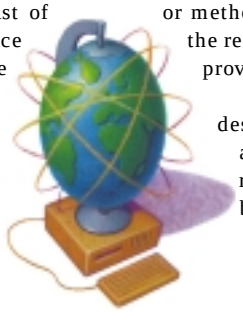
Enterprise business logic, however, is well suited to component-based design and its implicit decoupling of system and application functionality. System functionality, including security, transaction and life-cycle issues, are managed independently of business logic invocations. EJB components don't rely on request and response types or stacks of user-crafted filters to define their environmental functionality; instead, they rely on their container for such things, and the business logic remains separate.

Either one of these models alone won't fit the bill. In a complex enterprise application, we require connected subsystems that follow both models. In most real-world cases this means pulling logic out of JSP/servlets and out of any dependent proprietary libraries they may rely upon.

EJB Design Best Practices

Once convinced your application would benefit from EJB usage, how do you go about making the change? Refactoring, or methodically improving the design of existing code, is the recommended approach, and EJB design principles provide the engineering ammunition.

There are several emerging strategies for EJB design, covering topics from the granularity of data access to inheritance to caching. At a high level, refer to the following design principles and EJB best practices:



Principles

- Session EJBs are extensions of the client into the server, and should be used for any logic that the JSP/servlet application previously executed.
- Entity EJBs are representations of data that exists in an underlying datastore, and should be used to encapsulate logical groups of data (such as a customer, a company, etc.) as well as any logic inherent in that data collection. For example, include logic that returns different values depending on whether a specific customer entity is a regular customer or special customer.
- Resource factories provide a portable means for accessing subsystems such as mail, messaging, and the like. Use resource factories through JNDI rather than directly accessing a resource through explicit delegation or hard-coded references.
- JavaBeans, whether exposed in JSP pages or encapsulated in taglibs or GUI elements, provide excellent places to store client-side references to remote EJBs.
- Client interaction with application logic and resources exists in a fluid, contextual data structure that may be transformed either explicitly or implicitly into EJB value objects, JavaBeans, files, or database tables. Even when not explicitly architected, this context exists in some form across all layers in an enterprise application.
- Consider the network and your hosting facilities while architecting your particular software. If you have one large database host but only a small Web and middleware host, consider moving much of your logic into stored procedures and calling them via JDBC in session beans. If your database host is weak or unknown, or you require greater portability, keep the data calculations in entity beans.
- Most J2EE server vendors provide optimization parameters for improving the overhead of EJB loading and storing. Use them. At minimum, these optimizations should follow the three commit options described in the EJB specification.
- Consider using a single stateless session bean to provide access to other EJBs (this is a façade pattern). All of your EJB clients request operations through this session bean rather than making calls and obtaining references to multiple EJBs. This not only satisfies aesthetics, it also optimizes multiple EJB references and calls by keeping them in process, useful for local beans in particular.
- Merely porting a large body of code into a few EJBs won't resolve anti-pattern problems (EJB is not a silver bullet). You should still segment business logic into sublayers, where each sublayer is composed of EJBs with different functions; for example, consider an access layer to present a public API, an invocation layer to handle references and lookups, a logic layer to handle application algorithms and an entity layer for data representation.

> "IN DESIGNING SOFTWARE, MOST OF US WILL AGREE THAT MIXING PRESENTATION LOGIC AND BUSINESS LOGIC **IS A NAUGHTY THING TO DO**"

Best Practices

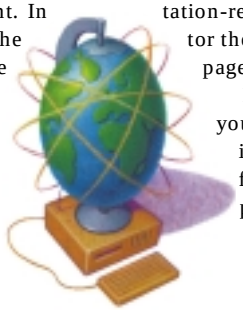
- Entity EJBs should be coarse-grained, with one entity representing a large type of data grouping rather than a number of smaller types. For example, craft one customer entity rather than separate entities for each type of customer.
- Entity EJBs should contain aggregate get/set methods that return chunks of data rather than fine-grained get/set methods for individual attributes. For example, instead of creating getStreet()/setStreet() methods along with getCity()/setCity() methods, create a single getAddress() method. This eliminates frivolous database, transactional, and network communication overhead.
- Decouple EJB access from your pages and servlets. At a minimum, use a tag in a taglib to call an EJB and execute its methods; also consider relying on a JavaBean decoupled from your taglib, where that JavaBean contains all the EJB access code.
- Avoid stateful session beans in large public enterprise applications (as opposed to intranet or small-distribution applications). Stateful session beans are resource-heavy, since one instance is maintained for each client. In Web-based applications, JavaBeans scoped to the user session or taglibs often provide reasonable lightweight alternatives to stateful session EJBs.
- Under heavy loads, entity beans should do more than merely represent a table in a database. If you are merely retrieving and updating data values, consider using JDBC within session beans instead.
- Monitor pattern sources, such as the J2EE Blueprints and the Server Side, for specific best practices, designs, and case studies applicable to your projects.

Refactoring Strategies

There are two general approaches to the actual JSP-to-EJB refactoring, and each is equally viable. The top-down approach involves starting with the presentation layer, while the bottom-up approach involves first sorting out the data access layer and its collections. The choice of strategy is largely a function of your personal engineering approach and the nature of the project at hand.

The top-down approach is common to Web-heavy applications and to developers employing an Extreme Programming methodology. This strategy asks you to start with your users and the presentation elements they see. Take a look at all the Java code in your JSPs. Pull that code out and group related functions into methods. Stash these methods in tag libraries. If you are using a proprietary code library, identify the presentation-related methods in your library's classes and refactor them into a taglib. Replace the Java code in your JSP pages with these tags.

Next, identify the heavier business logic that is in your taglib and/or proprietary library, and move it into session EJBs. The easiest way to do this is to find the relevant methods and simply cut-and-paste them into a session bean implementation.



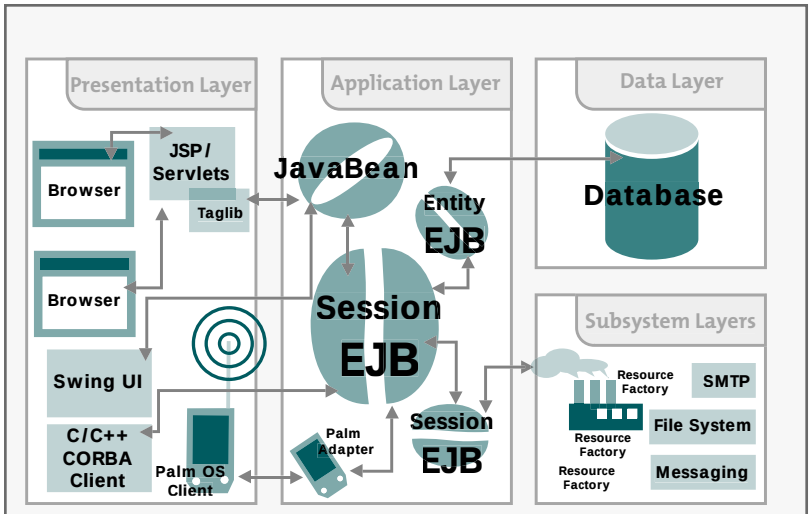


FIGURE 3 Refactored EJB version of the application

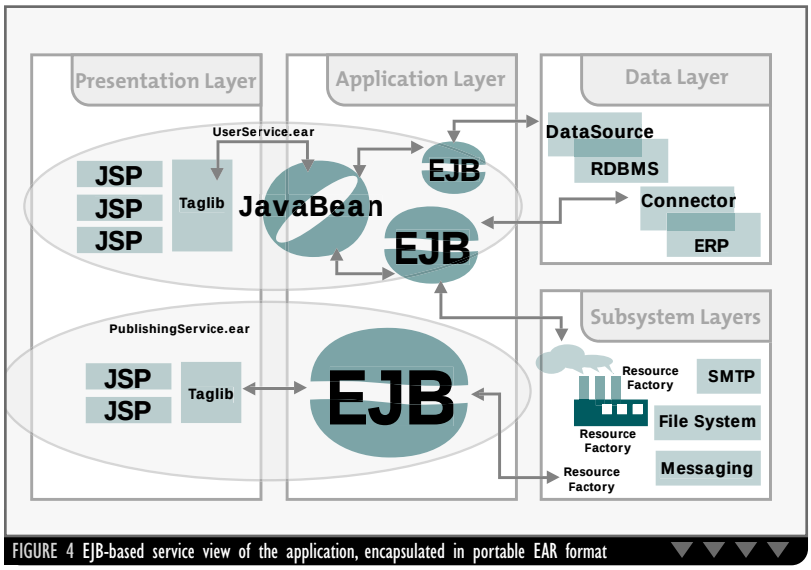


FIGURE 4 EJB-based service view of the application, encapsulated in portable EAR format

Don't worry about writing the bean's remote interface first. Since you have existing legacy code, you are refactoring rather than writing an EJB from scratch. After you've moved the business logic into session bean methods, write the bean's remote and home interfaces to abstract those methods. Finally, identify the core-data components and pull them out of the session EJBs and into entity EJBs. More on those in a moment.

The bottom-up approach sets the Web site aside while first delving into the core-data requirements. The strategy for top-down refactoring applies in reverse. Pull the data access and collection code out of JSPs and taglibs and move them into entity EJBs. Got a Hashtable called *customer*? Sounds like an entity EJB to me. Then move the entity EJB accessor code (that is, the client code) out of the JSPs and their taglibs and into session beans. Finally, build any new tag libraries that you require for presentation layer transformations. The idea is that JSPs use taglibs or servlets to call session beans (or MessageDriven beans for asynchronous functionality) and session beans may call other session beans or entity beans in order to execute logic for the tags. Figure 3 illustrates this scenario, in which the components contain only segmented application logic. The components delegate system logic to their containers.

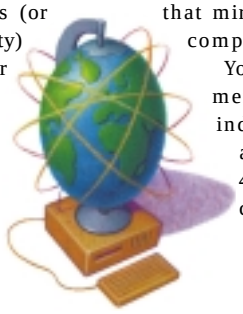
Regardless of whether you pursue a top-down or bottom-up strategy, the most time-consuming element of this process will likely be refactoring your data access and data manipulation code into entity EJBs. In general, it's advisable to use Container Managed Persistence (CMP) entities rather than the Bean Managed Persistence (BMP) alternative. But again, you have legacy code; don't be afraid to work against the grain of standard practices.

If you already have JDBC code in your servlets, taglibs or session beans, first create a BMP entity. Work first with the bean implementation, skipping the remote and home interface creation for now. This approach means essentially cutting-and-pasting the JDBC code into the appropriate entity life-cycle methods. After you have the implementation, then write the interfaces (the reverse of the normal procedure). Creating a BMP entity EJB from existing JDBC code in this manner is extremely quick, and after you've fashioned such a BMP entity, you can further refine it to make use of CMP, which typically provides better performance (due to data caching) in addition to its obvious portability.

Another advantage CMP offers is that the EJB is independent of any particular database and datastore type. It may rely on a relational database, but it may just as easily rely on SAP or some other ERP system with no changes to the bean code. The EJB need not specifically worry about such things, and is therefore portable (a very attractive feature of EJBs). EJB QL, the datastore-agnostic query language for CMP entities, standardizes this abstraction syntactically under the EJB 2.0 specification. In most vendor implementations today, regardless of whether they support EJB QL, transforming a BMP entity into a CMP entity means moving the SQL code out of the JDBC routines and into some object-relational mapping facility, such as a vendor-specific XML file.

Refactoring isn't complete without a repackaging of your application according to its functional units. Enterprise applications consist of JSP, servlet, and taglib files collected into Web archives (WARs) and EJBs collected in Java archives (JARs). The format for this collected application is the Enterprise Archive (EAR). Think of an EAR as a molecule, where the JARs and the WARs are the molecule's atoms. The metaphor extends further to compare the EJB's internal pieces to subatomic particles. This is no mistake – J2EE apps follow a deliberate component-centric design that mirrors beauty in nature. Components consist of components.

You can mirror this in your packaging. Instead of merely deploying the layers of your application individually (a Stovepipe process), package them as a logical organic service (as shown in Figure 4), which can be deployed on any certifiably compliant J2EE server.



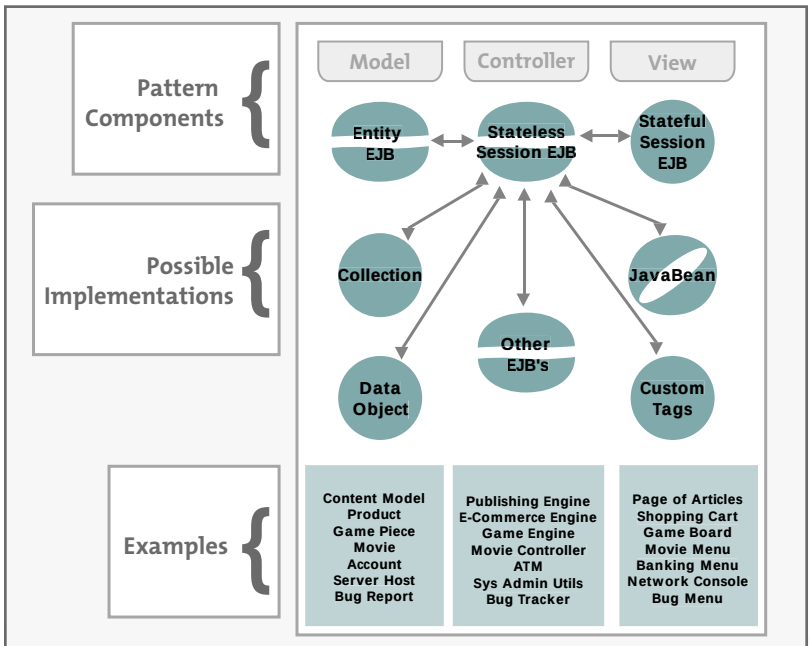


FIGURE 5 MVC view of enterprise applications

Applying MVC to Enterprise Services

Three basic categories exist in the software patterns community: architectural patterns that define high-level frameworks (such as the layer pattern used throughout this article's illustrations), design patterns that outline system and subsystem strategies, and idioms that describe language-specific tactics. Among architectural patterns, few have garnered as much attention as the MVC pattern, which originated in Smalltalk user-interface design. The idea is that the presentation View is cleanly separated from the data Model that it presents, and that both data and presentation are refereed by a Controller containing business logic. Figure 5 presents some examples of MVC implementations, in which a Controller EJB sends requests for logic to other EJBs in order to create or transform the Model, which represents the data. The Controller returns relevant pieces of that data to the View, which prepares it for client-specific display.

The Controller fits snugly into the application logic layer of our enterprise service. The central Controller object does not necessarily execute the business logic itself, but it's at least the entry point into the business logic. A stateless session bean is a good fit, as it scales well and can efficiently pass off requests for heavy logic to other EJBs. This Controller session bean could be accessed directly by tags in a tag library, or by a servlet.

The Controller is the common BLOB culprit in failed MVC implementations. Often when logic is removed from the presentation tier, the Controller grows fat and convoluted as a result. To avoid this, follow the aforementioned best practices, and segment the Controller into sublayers. Then apply proven design patterns to each of these segments.

The WAR component of an application, which includes the JSPs and taglibs, is the View of a Web-based service. If you find business logic winding its way into these elements, then you're flirting with violating strict MVC.

The Controller updates the Model in accordance with messages received from the JSP-based View. The Model may be a heavyweight entity EJB, or value object, or a lightweight data collection that the Controller sends as a parameter to EJB method invocations. Often the Model is all of these things at different periods of its life cycle; as the Model passes through layers it may be transformed into any number of implementation constructs. Through method invocations, the Controller transforms the Model and returns the Model to the Controller. Whether the Model is best represented as an entity bean, collection, or value object is a complicated design decision that's dependent upon the size of the data structure, the transactional attributes associated with it, and the number of method calls necessary to manipulate it. Multiple invocations on remote objects such as entity beans result in high network usage, but large data objects are much bulkier to send across the wire than entity EJB stubs. Let your project analysis determine the correct implementation.

In the end, users view pages, which are created by designer-crafted JSPs and their developer-crafted custom tag libraries. As users navigate through content, the taglibs send messages to the developer-crafted Controller code. This Controller translates the user's actions into requests for application logic; the Controller executes the logic by calling various developer-crafted EJBs. In this scenario, all of the EJBs are part of the Controller. In a long-lived session, the Model isn't re-created but continuously transformed by the Controller EJBs and remembered between calls by the View. The Controller sends pieces of this Model (or even the entire Model) back to the taglibs for translation into presentable elements on a JSP page.

The Future of EJB-Based Services

Once JSP-based applications are refactored into independent but connected services, the app is prepared for swift evolution. As vendors create new resource adapters, developers add entirely new subsystems to apps without interfering with any existing functionality; obsolete logic is removed without interfering with the Web presentation; new EJB components are purchased, removed, and added to existing services; entire applications are tied together and ported across unrelated enterprises; legacy data in mainframe systems filter effortlessly into existing services. The possibilities are powerful when the design and implementation are atomically elegant.

Resources

The definitive guide to refactoring strategies, complete with a catalog of refactoring patterns and idioms, is Martin Fowler's *Refactoring: Improving the Design of Existing Code* (Addison-Wesley, 1999). Strong EJB books are difficult to come by, but Ed Roman's *Mastering Enterprise JavaBeans* is a good resource, and its latest incarnation is available for public review at www.theserverside.com. That site and www.javasuccess.com are both good online sources for advanced EJB-specific design and patterns.

Author Bio

Patrick Sean Neville is principal engineer at Macromedia, where he is the architect of JRun's EJB server, core-service infrastructure, and other enterprise components. Prior to joining Allaire/Macromedia, he designed and implemented distributed applications for companies ranging from media agencies and Web start-ups to banks and financial institutions across the country. He is the creator of the Code Studio (www.codestudio.com), a supporter of open-source endeavors, and a frequent author and speaker on J2EE topics.

sneville@macromedia.com

ServletExec

Web Application Server by New Atlanta

WRITTEN BY JIM MILBERY

AUTHOR BIO

Jim Milbery is a software consultant based in Easton, Pennsylvania, with Kuromaku Partners LLC. He has over 17 years of experience in application development and relational databases. He is the applications editor of Wireless Business & Technology*, the product review editor of* Java Developer's Journal*, and the author of* Making the Technical Sale*. Jim can be reached via the company Web site at http://www.kuromaku.com.*

jmilbery@kuromaku.com

New Atlanta Communications, LLC
1041-D Cambridge Square
Alpharetta, Georgia 30004-1871
Web: www.newatlanta.com
e-mail: info@newatlanta.com
Phone: 678 366-3211

Test Environment
Dell 1400 PowerEdge 1 CPU (795MHz), Windows 2000 Server SP1 256MB RAM

In a recent editorial meeting with the ***JDJ*** staff I broached the subject of open source software with editor-in-chief, Alan Williamson. I freely admit that I was baiting him – but Alan was favorable towards it. It's a touchy subject for any technical person, sort of a motherhood and apple-pie thing.

Well, here it is folks, I've spent much of my technical career working for software vendors and I'm not so foolish as to bite the hand that has fed me in the past. The old adage that you get what you pay for is as true as it ever was. Before you start flaming me, let me just say that I'm not an "Open Source Luddite" and I freely admit to using the occasional open source product (Web browser, e-mail client, and HTTP server). In general, I prefer to use commercial grade products for everything else. Thus, I was more than happy to dump my copy of Tomcat into the virtual shredder and try out New Atlanta's much-vaunted ServletExec 3.1 Application Server.

New Atlanta makes the ServletExec product available for download from their Web site. It comes packaged in a 3MB InstallShield download kit for Windows (and shell scripts for UNIX). ServletExec can be installed as both a separate application server (Apache, Microsoft IIS, iPlanet Enterprise Web Server), or as an in-process Servlet engine (IIS/iPlanet). The installation expects you to have a JDK preinstalled on the server, but otherwise it's a breeze. The ServletExec developers jumped on the Servlets bandwagon way back in 1997, and their technical acumen shines through in the product.

I chose to install ServletExec as an application server and connect it to my Apache Server. New Atlanta didn't support the Apache release that I was using, so I dropped back to 1.3.14 in order to proceed with my tests. Once I had the correct version of Apache running, the ServletExec install went like clockwork. It correctly configured ServletExec to service Servlets and JSPs on behalf of my Apache Server. In fact, you can even run the installer on an ongoing basis to move Web application definitions from the ServletExec configuration over to the Web server's configuration files. I'd like to see this clever little configuration program uncoupled from the installation program and offered as a separate utility.

I'd have to say that ServletExec was one of the easiest products to install and configure in the application server category. The panels were easy to follow and the installation didn't wipe out any of my preexisting Apache settings (as some other products have done). The postinstallation process is also a breeze. ServletExec's management console (as shown in Figure 1) provides access to all the various capabilities of the server. You can control the depth and content of server logs, change the JVM, and configure Web applications (WARs) directly from the browser interface.

I was able to create a couple of quick JSP pages, package them into a WAR file, and create my own Web application in a matter of minutes. Experienced Java Servlet developers will find that they can control lots of detailed settings within the ServletExec environment, including servlet chains, servlet aliases, and the ability to define multiple logical "servers" within a single server instance. You can create separate ServletExec instances and tie them to different applications – even on the same machine. Thus, it's a simple process to host applications for multiple audiences on the same set of hardware.

You have complete control over the session-tracking environment, including the ability to control session persistence across server restarts. New Atlanta also provides lots of sample servlets. My only real complaint is that ServletExec doesn't provide any direct support for JDBC. If you want connection pooling or JDBC drivers, you'll have to use some additional third-party solutions. (ServletExec 4.0 will provide support for managing JDBC resources.)

Summary

ServletExec 3.1 is a compelling product for those organizations that want to build dynamic Web sites and applications – without the cost and overhead of an enterprise-class application server. Keep a sharp eye on New Atlanta. They recently cancelled plans to merge with Unify Corporation, and it's my guess that the company will be hitting the market with renewed vigor as a result of this change in strategy.

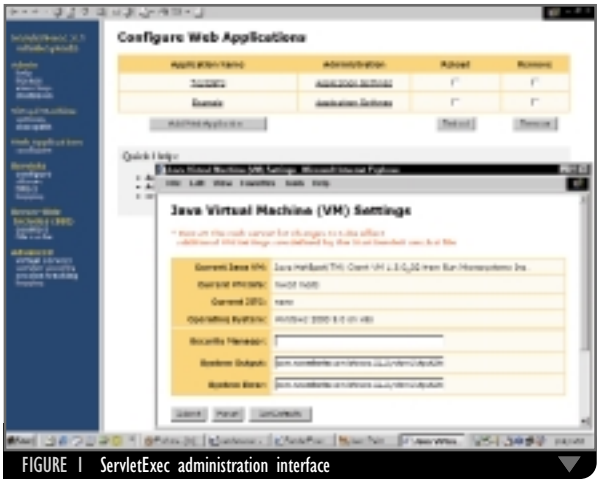


FIGURE 1 ServletExec administration interface

Using the VisualAge for Java Enterprise Access Builder for Transactions

WRITTEN BY
BRADY FLOWERS

Last month in *JDJ* (Vol. 6, issue 5) we looked at the Java 2 Platform, Enterprise Edition (J2EE) connector architecture (JCA) and its common client interface (CCI). To recap, JCA is the part of the J2EE 1.3 specification that facilitates the integration of Java applications with Enterprise Information Systems (EISs). The term EIS refers to a number of systems such as ERP, legacy databases, or transaction processing systems.

As we saw in the last issue, integrating these systems into a modern e-business architecture is complicated, to say the least. The APIs that these systems provide, if they provide one at all, are in general proprietary and require the developer to reinvent the process for each and every system that needs integration. JCA is Sun's proposed solution to this problem. It should do for EIS integration what JDBC did for relational database access.

EAB and CCF

As the vendor of CICS, MQSeries, Encina, IMS, and Host on-Demand, IBM has a vested interest in EISs. Since 1998 they've delivered a framework for accessing them (and SAP R/3) in VisualAge for Java. That product's Enterprise Access Builder for Transactions (EAB) feature and the common connector framework (CCF) bear a striking resemblance to JCA and CCI and it's easy to see why; IBM is a member of the expert group conferring with Sun on JCA.

Both JCA and CCF define abstract objects for connections, interactions, transaction management, and input and output structures. They both assist the application server to better manage resources, such as connection pools for the EIS connections.

At present there's one big difference. JCA isn't formally released yet and it's not implemented by any application server or tool vendor. EAB and CCF are available now, robust and mature. If you need to integrate any of the EAB-supported EISs, you can leverage the power of this framework now. Of course, since JCA is still a draft specification, you may need to make some changes to your applications when the architecture is finalized.

The ADDER Example

To illustrate EAB we'll use it to develop a set of classes in VisualAge for Java. These classes will encapsulate a single back-end transaction. For simplicity, we'll use a sample CICS transaction that comes bundled with VisualAge for Java. If you have a CICS host available, you can actually compile and link the transaction into the CICS subsystem to try it out.

The name of the CICS program is ADDER. It accepts two integer values for input and returns their sum. The COBOL source, located in the file <VAJavaRoot>\eab\samples\com\ibm\ivj\examples\eab\adder\adder.ccp, defines this record structure:

```
01  DFHCOMMAREA.
02  op1     PIC S99999 DISPLAY.
02  op2     PIC S99999 DISPLAY.
02  res     PIC S99999 DISPLAY.
02  keyNum PIC X(5) DISPLAY.
```

To make sure that the required features are installed into your workspace, start VisualAge for Java and press F2 to open the Quick Start dialog. Select "Features->Add Feature" and then select "IBM Enterprise Access Builder Library" and "CICS Connector." If you don't see one or both of these, the feature is already installed and you can move to the next step. If you're using a different example, make sure you install the connector for your EIS.

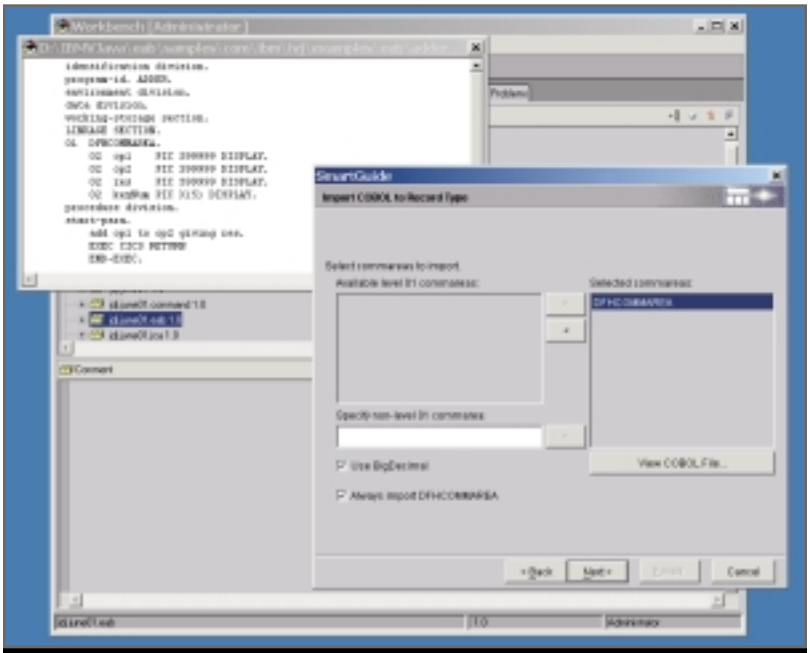


FIGURE 1 Import a COBOL file into EAB

AdderRecord

Create a project and package for the generated code. I called my package jdj.june01.eab. From the package's pop-up menu select "Tools->Enterprise Access Builder->Import COBOL to Record Type." Figure 1 shows a page from this dialog. Use the Browse button to navigate to the adder.ccp file. On the last page of the dialog make sure that "Create record from record type" is selected, name the output class "AdderRecordType", and click Finish. This will generate your AdderRecordType and AdderRecord.

AdderCommand

To create the command bean, select "Tools->Enterprise Access Builder->Create Command". Name the class EABAdderCommand, select "Edit when finished", and click Finish. This creates the command class and immediately starts the Command Editor as shown in Figure 2.

We must specify connection information. Right-click on "Connector" and select "Add ConnectionSpec". Use the Browse button to list all available connection specs for the connectors you've installed. Select "CICSConnectionSpec". When you highlight CICSConnectionSpec in the top-right window, its properties appear in the bottom window. In the URL field, enter the name of the gateway used to connect to the server. In the CICSServer field enter the name of the CICS server.

To specify the interaction right-click on "Connector" and select "Add InteractionSpec". This time select "ECIInteractionSpec" and, in its properties, enter "ADDER" for the programName property.

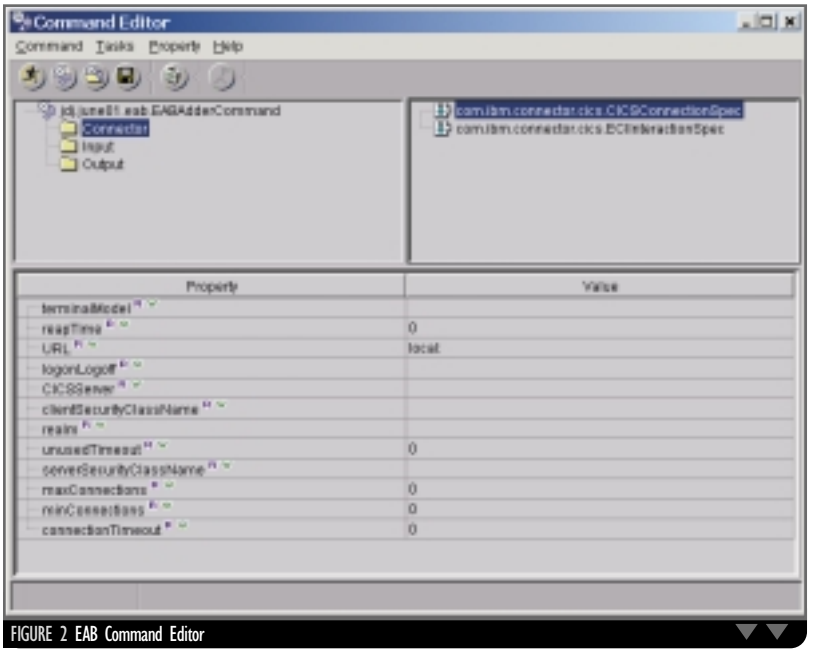


FIGURE 2 EAB Command Editor

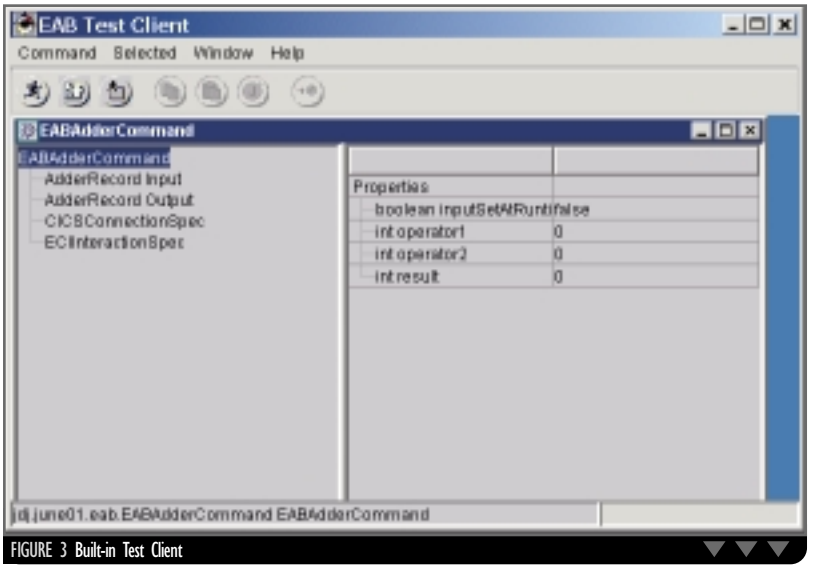


FIGURE 3 Built-in Test Client

Next we specify the input and output beans. Right-click on "Input" and select "Add Input Bean". Make sure "Implements IByteBuffer" is selected and use the "Browse" button to select AdderRecord. IByteBuffer is a special superclass that knows how to parse COBOL COMMAREAs and other byte-buffer types of data records. Specify an AdderRecord for "Output" as well. Now, select the input bean from the upper-right pane so its properties appear in the bottom pane. Right-click the "op1" property and select "Promote property". This causes the code generator to create getOp1() and setOp1() methods, which delegate to the input bean methods. Likewise promote the "op2" property of the input bean and the "res" property of the output bean.

Test the EAB Command

Normally we'd have to write a class to test our code. The Enterprise Access Builder relieves us of this task, not to mention the chore of debugging the test code. It has a built-in test client that we can execute from the Command Editor or from the Enterprise Access Builder menu. After launching the test client, select "Command->Create new instance" to get a list of all classes in our workspace that inherit from com.ibm.ivj.eab.command.CommunicationCommand. Pick EABAdderCommand from the list. The test client instantiates the class and displays the dialog you see in Figure 3. From here you can inspect and adjust any of the properties of EABAdderCommand. Set the inputs op1 and op2 to some values and invoke the command from the "Selected" menu. Assuming all goes well and you have your CICS gateway set up correctly, you should see the results in "res".

After only a few minutes of work and with no handwritten code, we now have a command object that can be used from client Java code with just a couple lines of setup:

```
EABAdderCommand cmd = new
EABAdderCommand();
cmd.getCeConnectionSpec().set... //
set any required properties
cmd.getCeInteractionSpec().set...
// set any required properties
cmd.setOp1(2);
cmd.setOp2(3);
try {
 cmd.execute();
 System.out.println("Result = " +
cmd.getRes());
}
catch (Exception e) {}
```

EAB Is Not JCA

What can we do now to leverage the power of EAB and still be able to migrate to a JCA implementation later? In addition to different class and package names, we have architectural and philosophical differences that are nontrivial:

- JCA connections are created by a ConnectionFactory that was obtained via a JNDI lookup, while EAB handles connections via the com.ibm.connector.Communication class.
- The JCA interaction behaves like a combination of the EAB interaction and the EAB command.
- In JCA, local transactions are accessed and controlled through a LocalTransaction object obtained from the connection; in EAB the RuntimeContext handles transactional boundaries and scope.
- Unlike EAB, JCA has the notion of result sets.

Reprising a Command Performance

We could write all the code to implement the CCI classes with the EAB code providing the "plumbing." We could, but we won't because:

1. It's a lot of work and hard to do, not to mention hard to do correctly.
2. JCA isn't finalized; our code may become obsolete at any moment.
3. The vendor or another party will provide a cleaner, more robust JCA implementation for us sooner or later.
4. Did I mention it's a lot of work and hard to do?

One solution would be to reuse the notion of command. The command pattern characterizes any activity by wrapping it in a straightforward interface. We have "setters" for input values to the command, an execute() method, and "getters" for resulting values.

In the example code (listings are located on the ***JDJ*** Web site) we've defined a command interface and an associated CommandException class in its own package. The command interface is intended to be generic, to serve as a front end for both our EAB-generated classes and any future JCA-based implementations. We'll address only those features we need right now: the ability to set input parameters, to execute the EIS transaction, to read output results, and to handle exceptions if they occur. Our design is generic so we can add other JCA features when we need them.

The command interface and CommandException are located in the jdj.june01.command package in the code listings. We've written jdj.jun01.eab.AdderCommand to implement the command interface. It contains an EABAdderCommand to do the work and "promotes" the relevant properties of the input and output records, making them accessible to the client code, and implements the execute() method and exception handling. We've included a main() method for testing; all you need to do is fill in the server properties for your system and execute it.

Conclusion

When it's time to make our sample work with a JCA infrastructure, we need to create only an AdderCommand that uses JCA. The client code won't have to change. In the package jdj.june01jca there's an AdderCommand that shows what this would look like.

Resources

1. *The JCA specification and information on CCI:* http://java.sun.com/j2ee/connector/
2. *IBM documentation for the Enterprise Access Builder for Transactions:* www7.software.ibm.com/vad.nsf/Data/Document3852
3. Gamma, E., Helm, R., Johnson, R., and Vlissides, J. (1995). *Design Patterns.* Addison-Wesley.

Author Bio

Brady Flowers is a software IT architect with IBM's WebSpeed team specializing in WebSphere, Java, and the rest of IBM's suite of e-business applications.

bradenf@us.ibm.com

iPlanet

New Web and Application Server Family

WRITTEN BY JIM MILBERY

AUTHOR BIO

Jim Milbery is a software consultant based in Easton, Pennsylvania, with Kuromaku Partners LLC. He has over 17 years of experience in application development and relational databases. He is the applications editor of **Wireless Business & Technology**, *the product reviews editor of* **Java Developer's Journal,** *and the author of* Making the Technical Sale. *Jim can be reached via the company Web site at www.kuromaku.com.*

apps@sys-con.com

iPlanet recently announced some updates to their family of Web and application server products that help consolidate the overall product line. The new product positioning won't surprise iPlanet enthusiasts as it positions iPlanet as a superior solution for a broader range of the application server marketplace. The core of these announcements is a new release of the Web server software and new packaging for the application server lineup:

- iPlanet Web Server, Enterprise Edition 6.0
- iPlanet Application Server, Standard Edition 6.0
- iPlanet Application Server, Enterprise Edition 6.0
- iPlanet Application Server, Enterprise Pro Edition 6.0

Although iPlanet is separate to some degree from Sun Microsystems, the two companies are very closely aligned. The new product lineup reflects this partnership in several ways. First, the new product line is an essential component of the Sun Open Net Environment (ONE), which is Sun's initiative in the Web Services arena. The product suite has always shared a common infrastructure, and the new product family has been designed to improve on this integration. The updated product suite is targeted toward a diverse audience:

- Service providers and enterprises looking to create dynamic content
- Customers looking for an entry-level Java 2, Enterprise Edition (J2EE) platform
- Core J2EE customers who want to leverage the full power of the J2EE platform
- Customers needing to tie legacy applications together to automate a business process

The iPlanet Web Server has long had a strong presence in the corporate marketplace. Although the Apache HTTP Server rates the most Internet hits according to Netcraft, the iPlanet Web Server has a much stronger following within the enterprise market. iPlanet's new packaging reflects this enthusiasm within the corporate environment and provides their customers with an integrated stack of products that follow a simplified growth path. The iPlanet Web Server Enterprise Edition 6.0 supports both servlets and JavaServer Pages, allowing customers to deliver dynamic Web content alongside existing static HTML pages using a single server. When it comes time to add support for advanced EJB capabilities, you can easily add the iPlanet Application Server to the mix.

The iPlanet Application Server comes in three versions, so you only need to pay for the specific features that your organization requires. Advanced servlet/JSP applications can be built with the Standard Edition, which includes native support for Oracle, IBM DB2, Sybase, and Informix. The Enterprise Edition adds support for EJB deployment (including EJB failover capabilities), providing developers with the full power of the J2EE platform.

Should you need to add process management and application-to-application integration to the mix, iPlanet offers the Enterprise Pro Edition. The upgrade path between the various versions is almost transparent, so it's a simple matter to start small and add to your applications as needed. The Forte for Java development environment works with all three editions of the Application Server lineup – so your developers can work with all three versions using a single IDE.

In fact, iPlanet has launched an aggressive campaign to woo developers to the iPlanet platform. In the past, developers had to join their developer program and pay a $995 registration fee to gain access to the application server product line. However, you can now download trial versions of the Enterprise Edition from the iPlanet Web site free of charge.

Should you wish to develop an application on the iPlanet Server, iPlanet has put together an aggressive developer package in the form of the iPlanet Developer Pack. The iPlanet Developer Pack, Enterprise Edition 6.0, improves developer productivity by combining a development license of the iPlanet Application Server, Forte for Java Internet Edition 2.0 trial software, and WebGain Studio 4.1 trial software – all for $1,295 per developer. The developer pack includes access to over 120 prebuilt EJB components from a variety of iPlanet partners, including ComponentSource, Compoze Software, Diamelle Technologies, Evergreen Internet, Flashline, and Omix.

Many of the organizations that I talk with are developing portal strategies for both internal and external access. iPlanet now offers the iPlanet Portal Server, which is built on the iPlanet Application Server platform. This Portal Server addresses all the critical portal requirements, including single sign-on, personalization, application integration, and knowledge management. Most organizations start their portal efforts with simple e-mail and calendar access and this is where iPlanet really shines, with its integration with both the Messaging and the Calendar Server.

iPlanet's product suites are easy to install and maintain, but Sun and iPlanet have also announced a set of preconfigured hardware/software packages to make the entire process even easier. Each solution features the iPlanet Web Server, the J2EE-certified iPlanet Application Server, and the Solaris Operating Environment 8.0 combined with various configurations of Sun server hardware built around the 64-bit SPARC microprocessor architecture. These recent announcements and product upgrades make a compelling story and have "upped the ante" in the hotly contested Java application server market.

JAVA DEVELOPER'S JOURNAL JDJ WORLD CLASS AWARD

iPlanet E-Commerce Solutions
World Headquarters
901 San Antonio Road
Palo Alto, CA 94303
Phone: 650 960-1300
Web: www.iPlanet.com
E-mail: see Web site

The Impact of EJB 2.0

An introduction to many of the issues that developers will face immediately

Last summer, Sun Microsystems released the first public draft of the EJB 2.0 specification with a lot of fanfare. Since then, it's been through a whirlwind of discussion, controversy, and modifications. Yes, modifications. The latest release of the EJB specification is Public Final Draft 2, which was released at the end of April.

WRITTEN BY
TYLER JEWELL

The latest incarnation of the EJB specification has a variety of features that developers should become familiar with:

- The introduction of message-driven beans
- The creation of a new entity EJB container-managed persistence model
- A model for creating container-managed relationships between entity EJBs
- The creation of a standard query language, EJB-QL, for querying EJBs and their properties
- The introduction of local interfaces for session and entity beans
- The renaming of remote interfaces to *component* interfaces
- The introduction of home methods
- The introduction of ejbSelect methods that allow an entity EJB to internally query for properties using a deployment descriptor–defined EJB-QL query

This article provides a rapid-fire implementation primer to the new features of the EJB 2.0 specification for developers with prior EJB experience.

The examples in this article are based on the simple UML diagram presented in Figure 1. Our mini e-business system will model components that might be used by a magazine publisher. Our magazine component will be a CMP-entity EJB that can have zero or more article CMP-entity EJB components in a one-to-many container-managed relationship (CMR). The client application will be able to query the magazine component for magazines and articles. Additionally, the client application will be able to register a subscriber to a magazine by sending a JMS message to a queue that has a registration message-driven bean consumer. The registration message-driven bean will simulate registering the subscriber and sending an e-mail message to confirm the information.

The examples provided with this article are configured to run on BEA WebLogic Server 6.0, Service Pack 1. The examples are based on the EJB 2.0 specification, Public Draft 1 (PD1). The examples do not support PD2 semantics since a vendor implementation supporting PD2 did not exist at the time this article was written. PD2 introduces requirements around the use of local interfaces for entity EJBs participating in a CMR relationship, a modified EJB-QL language, and a different DTD for the EJB deployment descriptor.

Message-Driven Beans

Message-driven beans are a new component type in the EJB 2.0 specification that allows developers to meld JMS with a component model. A message-driven bean is an EJB component that's a JMS consumer. Message-driven beans have all of the benefits of decoupled, asynchronous message-based models. For a good description of the benefits of using asynchronous invocations, check out the "Message-Driven Beans" chapter in the upcoming *Mastering EJB II* book that's undergoing community review at theserverside.com. Once the chapter has completed community review and is published, it will be made available for download in a PDF format.

Writing a message-driven bean is straightforward. Message-driven beans do not have client-visible home or component interfaces since they're decoupled from message producers. Thus, a message-driven bean's implementation is a class that:

- Implements the javax.ejb.MessageDrivenBean interface (ejbRemove() and setMessageDrivenContext(…))
- Implements the javax.jms.MessageListener interface (onMessage(…))
- Provides an ejbCreate() implementation

Listing 1 shows the implementation of a registration bean. (All listings appear on the ***JDJ*** Web site.) It contains comments explaining the bean developer's responsibility for each message-driven bean method implementation. In the registration bean, the onMessage(…) method expects to consume a message of type MapMessage. The MapMessage will have two string instances contained within it: one with the name of the magazine that should be subscribed to and another with the e-mail address of the individual that wants the subscription.

The examples provided in this article contain a client test application, Client.java (not listed). This application tests all aspects of the EJBs deployed for this article. In particular, for the registra-

tion bean, the client application creates a transacted QueueSender instance, fills a MapMessage with the appropriate values, and sends the message to the SampleQueue deployed in the application server. The registration bean's deployment descriptor contains the name of the JMS destination that the bean subscribes to.

Creating a deployment descriptor for a message-driven bean is equally simple. The DTD for EJBs has been extended to include a <message-driven> tag that a bean developer adds when the bean is first built. A message-driven bean is required to declare:

- ***The recommended JMS destination type for this bean.*** The recommended destination type can be javax.jms.Queue or javax.jms.Topic. These are recommendations made by the bean developer to the deployer indicating the type of destination this bean should consume messages from. This is declared with the <destination-type> tag. *Note:* the actual destination that a message-driven bean binds to is not listed in the ejb-jar.xml file. The destination that a message-driven bean binds to is application server–dependent and included in a vendor implementation deployment descriptor.
- ***If the recommended JMS destination type for a bean is javax.jms.Topic, the bean developer can configure the message-driven bean to be a durable subscriber of messages***. This allows a message-driven bean container to crash and later consume any missed messages. This is declared with the <subscription-durability> tag.
- ***An optional message selector to be used by the JMS consumer to filter out messages of interest.*** This is declared with the <message-selector> tag.
- ***An optional <acknowledge-mode> tag that tells a container using bean-managed transactions the semantics that should be used to determine when a consumed message should be acknowledged to the JMS server.*** The valid acknowledgment values are AUTO_ACKNOWLEDGE and DUPS_OK_ACKNOWLEDGE and have the same semantic behavior as listed in the JMS 1.0.2 specification.

Note: For message-driven beans that use container-managed transactions, the valid transaction attributes are Required and NotSupported. If the message-driven bean container is configured to use the Required attribute, message acknowledgment will be performed when the user transaction commits or denies when it rolls back. If the message-driven bean container is configured to use the NotSupported attribute, message acknowledgment is undefined in the specification but will likely happen after the completion of the onMessage(…) method.

Listing 2 contains the elements for all of the EJBs in this article. The <message-driven> tag declares the registration bean.

New CMP Model

The EJB 2.0 specification introduces a new format for handling container-managed persistence. It is commonly agreed that this new format is more flexible and easier to develop with than the EJB 1.1 model.

The specification provides a formal mechanism for declaring properties that should have their persistence managed by the container. This involves the bean developer altering his or her approach to developing an entity bean that will be managed through container-managed persistence. In particular, bean developers must now:

- ***Declare each property through a series of get/set accessor methods that are declared abstract.*** The data type of the property can be determined by a container generation utility by using introspection on the return type of the getXXX() method and the input parameter of the setXXX(…) method. Declaring CMP properties in the bean class is less ambiguous now since developers need only to provide a pair of JavaBean-like accessor methods in the implementation class.
- ***Declare the implementation class to be abstract.*** The application-server vendor will provide the concrete bean implementation class during container generation. The container generation utility will extend this abstract class with an implementation of its own that provides a concrete implementation for all of the abstract property accessor methods. The actual persistence logic for the EJB will be located in the concrete implementation created by the container generator.
- ***For each property declared with an abstract accessor method in the implementation class, the bean developer must provide a matching <cmp-field> declaration in the deployment descriptor.*** The value of the <cmp-field> tag must start with a lowercase letter and match the text following the get/set accessor methods in the implementation class. For example, if you declare <cmp-field>age</cmp-field>, the bean implementation class must have equivalent getAge() and setAge(…) methods.
- ***The primary key must be a CMP field in the bean implementation class with an appropriate <cmp-field> declaration in the deployment descriptor.*** Additionally, the <primkey-field> tag must be used to declare which <cmp-field> value is the primary key value for the EJB. Complex primary keys are still supported and have a slightly different semantic associated with them.
- ***The ejbCreate(…) method must return null.*** The concrete implementation will create an appropriate return value that will be handled by the container.

Listings 3–5 contain the implementation of a simple CMP entity EJB. The EJB has three CMP properties: articleID (primary key), text, and author. The implementation class declares each of these properties using the appropriate abstract accessor methods. Additionally, the class is declared to be abstract.

It's extremely important to understand that the container callback methods and the business logic implementation methods must use the abstract accessor methods to read or write any of the CMP properties. This is demonstrated in the ejbCreate(…) method where the properties of the bean are initialized using the appropriate set methods. Also

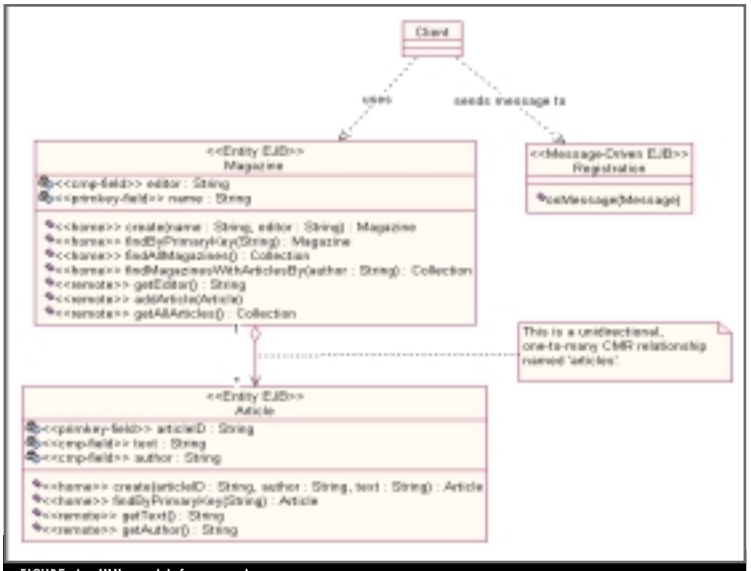


FIGURE 1 UML model for examples

> Container-managed relationships (CMR) are easy to define using the new EJB 2.0 model. **CMRs can be one-to-one, one-to-many, or many-to-many**

note that the ejbCreate(...) method returns null as defined by the specification.

Listing 2 contains the deployment descriptor for the article EJB. The declaration of the <cmp-field> and <primkey-field> tags are done for the article EJB as described by the rules listed above. You might be wondering what the intent of the <abstract-schema-name> tag is. The value of this tag will be used in any EJB-QL queries that are defined for your EJB. Its value acts as a symbolic data type that references this particular EJB in the query. Since multiple EJBs can be declared in a single deployment descriptor, it's required that each EJB has a unique <abstract-schema-name> value.

Because of the innovative approach taken to declaring CMP properties of an entity EJB, application server vendors can provide a whole series of enhancements to increase performance of entity EJBs:

- ***Lazy loading of fields when they're accessed***: The container can determine when a get method is invoked and load fields at the time of the invocation instead of pre-fetching values.
- ***Group loading of fields:*** An application server vendor can provide a way to declare groups of fields so that when one field is accessed, all of the fields for the group will be loaded as part of a single query.
- ***Optimized writes to a database:*** Since the container can monitor all get-and-set method invocations made, the container can optimize any write queries to write out only those fields that were modified, if any.
- ***The creation of read-only and read-write distributed data caches:*** These can be done at the container level in a much more automated way since properties can be monitored for alterations.

New EJB Query Language

Because of the declaration of CMP fields in a deployment descriptor, it's now possible to create queries in the deployment descriptor for any finder method other than findByPrimaryKey(). findByPrimaryKey()'s query is automated by the container. Queries are declared in the ejb-jar.xml deployment descriptor as part of the <query> tag. The <query> tag declares the method that the query belongs to, and the query itself.

EJB-QL queries can also be applied to ejbSelect() methods, which are new in the EJB 2.0 specification. ejbSelect() methods are private methods declared in an implementation class, and used by other implementation methods to access fields and collections of fields that are accessible from a particular EJB. This includes fields that are accessible through a container-managed relationship. Even though we do not provide an ejbSelect() implementation in our example, the definition of the EJB-QL query is similar to those for finder methods.

EJB-QL queries have three clauses: SELECT, FROM, and WHERE. The SELECT clause is needed for ejbSelect() method queries but can be dropped for finder queries. For example, Listing 6 contains the home interface for the magazine entity EJB that uses CMP for its name and editor fields and participates in a container-managed unidirectional, one-to-many relationship with article. Listing 2 contains an EJB-QL query for the findAllMagazines() method declared in the home interface. We want this method to return references to any magazine that exists in the persistent store.

Let's break down the value, <![CDATA[FROM MagazineBean mb WHERE mb.name IS NOT NULL]]>:

- ***<![CDATA[...]]>:*** This construct is used to escape any SGML tags that may appear within the query. The body of this declaration is the query.
- ***FROM MagazineBean mb:*** This declares a query variable, mb, of type MagazineBean. MagazineBean comes from the <abstract-schema-type> declaration of the EJB as mentioned above. The mb variable can be used to reference an instance of that particular EJB type in the WHERE clause of the query.
- ***WHERE mb.name IS NOT NULL:*** This clarifies the query. The dot '.' syntax is used to navigate to any CMP field or container-managed relationship. mb.name refers to the value of the name CMP field of the mb EJB instance. The dot '.' syntax can go many levels deep, but can not be used to navigate to a field or a relationship that references a collection of fields. A different syntax must be used.

This example will return references to all magazine EJBs whose name CMP field is not null.

New Container-Managed Relationship Model

Container-managed relationships (CMR) are easy to define using the new EJB 2.0 model. CMRs can be one-to-one, one-to-many, or many-to-many. Additionally, CMRs can be unidirectional or bidirectional. The bean developer is responsible for defining the relationships between beans in the implementation classes and the deployment descriptor.

The bean developer has to perform the following activities to define a CMR:

- Provide abstract accessor methods that indicate directionality and multiplicity in the implementation class. Similar to CMP field declaration, every CMR field must have a get/set method that references the other EJB in the relationship.
- If the target of the relationship has a multiplicity of one, then the get/set methods must reference the remote interface of the other EJB in the relationship. *Note:* In PD2, the specification requires accessor methods to reference only the *local* interface of the other EJB in the relationship.
- For relationships where the target of the relationship has a many multiplicity, the get/set accessor methods representing the relationship must use collection or set as their data type. Listings 6–8 contain a one-to-many, unidirectional relationship to the article EJB. The magazine EJB implementation class declares Collection getArticles() and setArticles(Collection) since the target of the relationship is a many multiplicity.
- The absence of get/set accessor methods in an implementation class indicates lack of directionality for that particular EJB. As a result, the article EJB implementation (Listing 5) does not have the get/set accessor methods to the magazine since this relationship is unidirectional.
- The source EJB, target EJB, directionality, and multiplicity of the CMR are

also defined in the deployment descriptor via a <relationships> tag that appears after all EJB declarations, but before the <assembly-descriptor>. The <ejb-relation> tag defines a single relationship between two EJBs. An <ejb-relation> tag has two <ejb-relationship-role> tags that define each end of the relationship. Each <ejb-relationship-role> tag defines the <multiplicity> for this role, the EJB that is the <role-source> of this role, and a <cmr-field> that indicates whether or not this EJB contains get/set accessor methods that implement directionality for this source.

Listing 2 contains the <relationships> tag for the magazine-article CMR. Notice that only the MagazineEJB source has a <cmr-field> declared since we're creating a unidirectional relationship. Additionally, the value of the <cmr-field> must match the text appearing after get/set in the accessor method.

It's important to understand that the actual persistent mapping of the relationship is not defined anywhere in the standard deployment descriptor; it is completely vendor-dependent. For example, the most common technique used to represent a one-to-many relationship in a relational model is to have a foreign key in one table pointing to a primary key in another. The foreign key would reside in the table representing the "many" multiplicity of the relationship and would point to the primary key of the table representing the "one" multiplicity. This abstract-to-concrete persistence mapping is not defined in the EJB specification and is left up to each EJB container vendor to supply. For BEA WebLogic Server, this is done in the weblogic-ejb-jar.xml and weblogic-cmp-rdbms-jar.xml files.

As a final note, it should be understood that EJB-QL is robust enough to be able to navigate CMRs. For example, the magazine EJB home–interface defined a findMagazinesWithArticlesBy(String) method that requires an EJB-QL query that navigates magazine and article. Since the dot '.' syntax can not be used to navigate relationships with a multiplicity of many, a special syntax has been created to support this. The query provided is FROM MagazineBean mb, article IN mb.articles WHERE article.author = ?1. Some points about this query include:

- The IN clause defines a loop variable that refers to a single instance defined in a collection. article IN mb.articles declares a local loop variable, article, that refers to a single instance of the collection of article references that are associated with this magazine instance as defined by the CMR field, articles.
- Since the article variable refers to a single instance of the collection, article can be used in the WHERE clause to reference CMP fields of the article EJB.
- ?1 is EJB-QL syntax for referencing the first input parameter of the finder method. ?2 would represent the value of the second input parameter, etc.
- The syntax for this example is based on PD1 and has changed slightly in PD2.

Conclusion

Whew! What a mouthful! As you can see, the EJB 2.0 specification made many modifications that impact EJB developers. This rapid-fire article attempted to provide a quick introduction to many of the issues that developers will face immediately. Now, it's time to go have some fun and implement the biggest and brightest e-business systems using your newfound EJB knowledge!

Author Bio

Tyler Jewell is a principal technology evangelist for BEA Systems, and an expert educator, mentor, and lecturer on enterprise technologies. He is a coauthor of Mastering EJB 2.0 *(Wiley) and a contributor to* Pure EJB *(SAMS). He is also a member of the O'Reilly Network's ONJava.com editorial advisory panel.*

tyler@bea.com

WRITTEN BY JEREMY GEELAN

Welcome to the Heart of Java

As Rich Green, vice president of Java software development at Sun and a contributor to this historic JavaOne issue of ***JDJ*** once said, J2SE technology "is the heart and soul of the Java 2 platform."

In other words, it allows software developers to build network-centric software that's compatible across all major platforms, helping to protect their customers' IT investments. The J2SE runtime environment supports the architecture of the J2EE platform, providing it – and the applications deployed on it – with all the standard Java facilities.

So for ***JDJ***, even in its new "2.0" incarnation, to be published without a section devoted to the Java 2 Standard Edition would be a little like the New York Times being published without its Op-Ed page, or a Ford Explorer being sent out of the factory without its wheels.

Accordingly, this month's selection of articles is plentiful and varied, ranging from John Goodson on Java database connectivity and Irvin Lustig on optimization to Blair Wyman on JNI programming in C/C++ and Michael Barbarelli's piece on using JavaBeans in Dreamweaver.

If you're reading this issue at Sun's JavaOne show, perhaps you've been saturated with the myriad possibilities of J2ME or maybe emboldened by the transformational powers of J2EE for e-commerce. There may not be too many sessions devoted to J2SE, any more than there are technical books with "J2SE" in the title available through Amazon.com (there aren't, not a one).

But J2SE isn't the "poor relation" of the Java industry any more than, say, paper is considered second fiddle when it comes to the publishing industry. Accordingly the J2SE Section will always give houseroom to good technical writing that explores the development of apps using the J2SE APIs. There will always be, just as there is this time, at least one book review (this time we've selected *Debugging Java* by Will D. Mitchell; next month we'll look at *3D User Interfaces with Java 3D* by Jon Barrilleaux). There will always be reviews of products and apps, and there will always be room, too, for an up-to-the-minute roadmap and for the J2SE FAQ, since ***JDJ*** is well aware of its duty to entrants to the Java space.

We'll print plenty of source code, plenty of tips and tricks...and of course want plenty of feedback.

Java programmers of every stripe should feel free to become involved with the ***JDJ*** Web site, www.sys-con.com/java, and should consider subscribing to JDJList, too, where Java information, insights, and viewpoints are exchanged on a daily, and often an hourly, basis.

Last but not least, as no reader of this issue could possibly be unaware – short of having read the issue through the wrong end of a telescope...or in the dark maybe – **SYS-CON Media**'s conference arm, **SYS-CON Events**, is producing and presenting in September what is simply the largest Java developer conference and expo ever held on the East Coast. So if you are able to get to New York anytime between September 24 and 26, make a firm note in your calendar to attend. Further details are also on our Web site; you can't miss them.

Java Developer's Journal undergirds the conference in much the same way that J2SE undergirds the Java 2 Platform. The new section editors will all be there, along with the head honcho himself, editor-in-chief Alan Williamson, and a good many of ***JDJ***'s stalwarts like product review editor Jim Milbery, plus an assortment of editorial panel members and regular contributors.

What's more, if you want to listen to the likes of the coinventor of Java, the legendary James Gosling, or to the man who helped orchestrate the growth and adoption of the Java platform from its infancy to its present status as a robust platform that supports mission-critical applications in almost 80% of Fortune 1000 companies – which is what conference keynote speaker Dr. Alan Baratz did in his former life as president of the software products and platforms division at Sun – then you should attend.

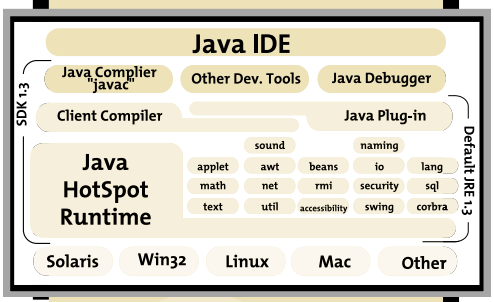
We look forward to seeing you there in September. Just as we look forward to meeting you here, in the J2SE Section of the new-look ***JDJ***, next month.

jeremy@sys-con.com

AUTHOR BIO

Jeremy Geelan, editorial director of SYS-CON Media, speaks, writes, and broadcasts about the future of Internet technology and about the business strategies appropriate to the convergence of business, i-tech, and the future.

J2SE ROADMAP



The Java2 Platform, Standard Edition defines the APIs for the core language.

J2SE

Javabeans

JavaSound

Java Native Interface

RMI-IIOP

Swing

Java2D

Java Accessibility API

Java Naming and Directory Interface (JNDI)

Java Foundation Classes

Useful URLs:
Java 2 Platform Standard Edition
http://java.sun.com/j2se/1.3/

Features:
http://java.sun.com/j2se/1.3/docs/relnotes/features.html

SPECIAL SECTION: J2SE FAQ's

What's new in J2SE?

J2SE has focused on adding new libraries. Some new features include the javax package, Swing, Java Collections API, JavaBean enhancements, and Java 2D. It also introduced a completely new security architecture and Java Hotspot.

At the same time J2SE was brought into existence, two other versions were split off – J2EE and J2ME.

What is Swing technology?

Swing is a set of GUI components with a "pluggable look and feel" based on the JDK 1.1 Lightweight UI Framework. The pluggable look and feel allows developers to implement an operating system–specific look and feel (Motif, Microsoft Windows, MacOS).

Swing components include both 100% Pure Java versions of the 1.1 AWT component set, plus a rich set of higher-level components. The complete set contains borders, buttons, checkboxes, combo boxes, icons, labels, lists, list boxes, menus, menubars, menu items, pop-up menus, radio buttons, progress bars, scroll panes and viewports, scrollbars, tabbed panes, tables, text areas, text components, text fields, trees, and HTML viewers.

What security enhancements are in J2SE?

J2SE introduces the concepts of *permission* and *policy*. When code is loaded, it's assigned permissions based on the security policy currently in effect.

These new concepts of permission and policy enable the Java Development Kit to offer fine-grained, highly configurable, flexible, and extensible access control. Now such access control can not only be specified for applets, but also for all code written in the Java programming language, including applications and beans.

What is Java 2D?

The Java 2D API is a set of classes for advanced 2D graphics and imaging. It encompasses line art, text, and images in a single comprehensive model. The API provides extensive support for image compositing and alpha-channel images, a set of classes to provide accurate color-space definition and conversion, and a rich set of display-oriented imaging operators.

These classes are provided as additions to the java.awt and java.awt.image packages (rather than as a separate package).

What about Java Collections API?

Collections Framework is a unified framework for representing and manipulating collections, thus allowing them to be manipulated independently of the details of their representation. It reduces programming effort, increases performance, allows for interoperability among unrelated APIs, reduces effort to design and learn new APIs, and fosters software reuse. The framework includes interfaces, implementations, and algorithms.

How have JavaBeans been improved?

Interaction with Applet Semantics eases the development of objects that implement applet as well as JavaBeans architecture behavior.

Better design-time support adds infrastructure during the JavaBeans component architecture design time to enable a more sophisticated builder environment.

Runtime containment and services protocol (BeanContext) adds additional context to the execution lifetime of JavaBeans architecture including, but not limited to AppletContext and the ability to parent into an AWT presentation hierarchy.

Building a Telephone/Voice Portal with Java Phonelets

Last month we introduced the phonelet architecture and developer API for creating lightweight telephone/voice services in Java. This article expands the automated answering machine with caller ID, basic voice processing capabilities, and, just for fun, a simple appliance control function using an inexpensive X10 power line module. We'll also introduce voice recognition and speech synthesis and provide recommendations for further exploration.

A how-to for both the newbie and the pro

Part 2

Written by Kent V. Klinner III & Dale B. Walker

Caller ID Makes Simple Phonelets More Useful

Caller ID, recently made available by most U.S. telephone companies, identifies the telephone number and sometimes the name associated with a particular telephone line. The information is transmitted between the first and second ring using Frequency Shift Keyed (FSK) modem tones. Most modems can be set to provide this information if it's on the telephone line.

Two formats are used for CID:

- ***Single Data Message Format (SDMF)***, which provides date, time, and calling number
- ***Multiple Data Message Format (MDMF)***, the more common format, which additionally provides the name associated with that number

Both formats, when provided by a modem, take the form of an ASCII hex string, where the first byte of the string indicates the message type.

Loox

www.loox.com

An MDMF string might look like this:

```
"80200108303332343039303207084A4F484E20444F45020A383030
353535313231327D"
```

This would decode to:

```
80: MDMF
20: 32 bytes of data
01 08 3033323430393032: date and time (length 8) 03/24
09:02
07 08 4A4F484E20444F45: name (length 8) JOHN DOE
02 0A 38303035353531323132: number (length 10) 800 555
1212
```

The CallerId Class takes such a string and decodes it into useful information.

Caller ID can transform a useless phonelet like HelloCaller into a useful application. The following code demonstrates the service method of a phonelet that never answers the phone, but always announces the phone number of the calling party.

```
public void service (PhoneCall call) throws
IOException {
 CallerId cid = call.getCallerID();
 String callerPhoneNumber = null;
 if (cid != null) callerPhoneNumber =
cid.getPhoneNumber();
 Talker.talk("Incoming call from
"+formatForReadback(callerPhoneNumber));
}
```

AnnounceCaller simply reads the phone number of the calling party to the PC user. The caller ID packet is available only on telephone lines that subscribe to the caller ID service of their local telephone company. If the packet is unavailable, the phonelet simply announces the caller as "unknown." The details of the announcement are implemented in the method formatForReadback(). The Talker.talk method assumes the presence of a text-to-speech package like IBM's ViaVoice. (See section "How to Handle Speech" for more detail.)

One of the most frustrating aspects of developing audio applications is wrestling with the wide variety of audio file formats, encoding schemes, sample rates, and resolutions. The narrow bandwidth and serial speed limits of voice modems narrow the selections for phonelet programmers, but there is still room for confusion. Table 1 illustrates the range of choices for some Lucent and Rockwell/Conexant chip sets.

The phonelets' framework uses javax.sound.sampled, and therefore supports wav file format and the encoding schemes defined for AudioFormat in Java v1.3 (PCM, u-law and a-law).

Three methods defined in the VoiceModem interface will help developers choose an appropriate audio format for playing or recording sound.

VoiceModem.getAudioFormats() returns an enumeration of audio formats supported by the modem. Developers who extend GenericVoiceModem to develop a device handler for a specific modem should specify at least one common audio format. Modem documentation can be difficult to obtain, but one of the most accessible sources of modem information is available in the INF files distributed with Microsoft Windows. Each modem manufacturer provides an INF file as part of the Windows installation package to specify the modem's operating capabilities and command sets. A description of modem INF files is beyond the scope of this article, but if you're familiar with AT command sets you won't have much trouble culling useful information for your model.

- ***VoiceModem.getAudioFormat()*** returns the currently active audio format.
- ***VoiceModem.setAudioFormat()*** sets the current audio format for the modem.
- ***VoiceModem.getPreferredAudioFormat()*** returns the preferred audio format for the modem.

The definition of *preferred* is up to the device handler developer. For example, some modems based on Rockwell/Conexant chips support a PCM 8-bit encoding at 11,025 Hz, but won't report DTMF events or silence intervals at that rate. At a lower sample rate of 8kHz the same modem will detect and report DTMF events as well as intervals of silence. Clearly, the higher rate would not be preferred for any application that needed to detect touchtones while playing a message for the user.

How to Play Audio

In the HelloCaller example we demonstrated how to play a wav file to a caller. We'll now discuss the Phone.play() method in more detail.

GenericVoiceModem doesn't provide on-the-fly audio format conversion. Developers must make sure that the audio format of the file, stream, or data buffer they intend to transmit with the Phone.play() method matches the currently active format of the phone. If you try to play an 8-bit/8kHz/u-law wav file through a phone set to 16-bit/7.2kHz/PCM, the result will be unintelligible. The following code illustrates the correct way to play a wav file. If the audio format of the wav file isn't supported by the phone's underlying modem, a ModemException is thrown.

```
try {
AudioFormat audioFormat =
AudioSystem.getAudioFileFormat(waveFile).getAudioFormat
();
phone.setAudioFormat(audioFormat);
phone.play(waveFile);
}
catch (PhoneException e)
{
 // thrown if audioFormat is not supported
}
```

ENCODING	BITS/SAMPLE	SAMPLES/SECOND	CHIPS
PCM	8	7200	Lucent, Rockwell
PCM	8	8000	Lucent
PCM	8	11025	Lucent, Rockwell
PCM	16	720	Lucent
A-Law	8	8000	Lucent
U-Law	8	8000	Lucent
ADPCM	2	7200	Rockwell
APCM	4	7200	Rockwell

TABLE | The range of choices for certain Lucent and Rockwell/Conexant chipsets.

The Phone.play() method allows callers to interrupt a message with a simple touchtone. This interrupt feature is especially useful when your phonelet is prompting the user for touchtone input and you want an instructional message to stop playing immediately upon detection of a specific touchtone key. In the following example we play a sequence of wav files, aborting the play if a certain key is pressed.

```
public void service (PhoneCall call) throws
IOException {
 Phone phone = call.answer(this);
 if (phone == null) return;
 phone.play("welcomeToCinemaPhone.wav");
phone.play("pressPoundToEndThisMessage.wav");

phone.play("theFeaturePresentationToday.wav",pressPound
           KeyToAbort);
 phone.play("thankYouForCallingCinemaPhone.wav");
 phone.hangup(this);
}
```

How to Record Audio

The service routine in Listing 1 demonstrates how to record a message from an incoming caller. In this example all instructional messages and prompts are played from wave files. (All listings can be found on www.JavaDevelopersJournal.com.)

A reasonable message recorder application should be tolerant of slow callers and resilient to premature hang-ups. Before a message is recorded, instructions should be repeated a reasonable number of times. If the caller doesn't respond with an appropriate touchtone or verbal response within a reasonable time frame, the phonelet should hang up. When recording commences, the prudent developer will make allowances for callers who hang up or are disconnected. Reasonable applications may also allow the caller to terminate the recording phase with a touchtone.

Phone.record() allows developers to limit recordings to a maximum time and terminate them when the device detects a defined period of silence or with a touchtone. The API documents the specific methods.

Not all modems will support silence detection. As mentioned previously, some Rockwell/Conexant chips won't report silence intervals or detect touchtones when the sampling rate is 11,025 Hz.

> "Voice recognition technologies have advanced rapidly in the last decade, **although speaker-independent voice recognition with large vocabularies remains the industry's holy grail**"

One of the design goals of the phonelets framework was to insulate the developer from the complexities of audio stream handling. The wide variety of hardware has made it almost impossible to hide all the details. Developers need to understand the basic capabilities of their hardware and, above all, be very paranoid.

The Phone.play() method accepts the touchtone string "one" as an interrupt argument. If the caller presses the "1" key while the instructions are playing, the phonelet interrupts the message and starts recording it.

The Phone.record() method accepts a maxTime argument and a maxSilence argument that stops the recording after a specified number of seconds or after a specified interval of silence has been detected (the caller may have hung up).

Finally, the phonelet plays a goodbye message to indicate that it is about to terminate the call. Of course, a really useful phonelet might do something with the message file before exiting, like e-mail it to the owner or copy it to a specific folder.

How to Place an Outgoing Call

Placing an outgoing voice call can be quite complicated for a modem. Data modems and fax machines answer with specific tones to signal their functions, but humans and answering machines aren't so predictable. Determining when a call has been answered has been a challenge for modem and telephony board manufacturers. The most common scheme among modem manufacturers is to set a timer that expires when the ring signal goes away. This method is crude and isn't always reliable.

Manufacturers have also had some difficulty determining precisely when a call has been terminated by the distant party. Some modems and telephony boards use silence detectors but, again, these methods may not work reliably, especially if the line is noisy. Other schemes have been developed but require a much more detailed discussion of telephony systems and audio signal processing.

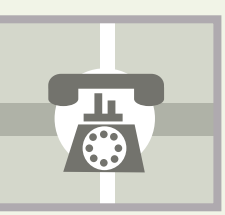
The phonelet framework allows developers to place outgoing calls with the Phone.call() method. It also allows developers to redirect audio to a specific OutputStream. We challenge readers to develop an audio stream processor that can accurately and reliably detect a spoken (or recorded) greeting. We also challenge you to develop a reliable adaptive silence detector.

Control Your World

A personal phone/voice portal can provide access to home and office appliances. Remote control can be easy with an inexpensive X10 serial line power control module. If you have a DSL modem in your home or office, you've probably experienced the kind of failure that requires you to cycle the power on the modem to reestablish your Internet connection. This kind of failure can be very inconvenient when you're away from your office and your Web server is stranded behind a stalled DSL connection.

With an inexpensive X10 serial port transmitter and a wireless power line control module, you can command a restart remotely with a telephone touchtone. You can purchase X10 appliance control modules at most electronics stores, but make sure you don't connect a sensitive electronic device like a DSL modem, a VCR, or a PC to a lamp dimmer module. Use the appropriate appliance module.

The Phonelet framework includes the X10Transmitter class to simplify the transmission of the control commands through an available serial port. A detailed explanation of X10 is beyond the scope of this article, but Listing 2 shows how simple it is to turn appliances on and off with touchtones.



How to Handle Speech

Voice recognition technologies have advanced rapidly in the last decade, although speaker-independent voice recognition with large vocabularies remains the industry's holy grail. All of the commercial voice portals employ speech recognition. The limited vocabulary of the navigation commands simplifies the speech recognition problem and results in a much more accurate, although not yet perfect, dialogue between caller and service. At least one service employs new hardware that accelerates the voice recognition algorithms with chips that cost as little as $2. Nevertheless, speech-driven navigation can be cumbersome and vulnerable to background and line noise.

Voice recognition is beyond the scope of this article, but the phonelets framework can be extended to support a third-party voice engine.

Text-to-speech systems have also improved dramatically in recent years. The robotic monotone of earlier engines is giving way to a much more natural elocution. Lernout and Hauspie, in particular, have produced a text-to-speech system that closely approximates the natural rhythms of the human voice. Nuance also provides a sophisticated and Java-compatible framework for text to speech. Speech synthesis is likely to be a much more useful addition to a lightweight voice portal than speech recognition as it'll allow callers to access information from home, office, and Web databases that would otherwise be available only via PC, wireless PDA, or Web-enabled cell phone.

The Java Speech API attempts to standardize the interface between applications and third-party speech products, but severe limitations in the current API will force service developers to use vendor-specific interfaces. Specifically, JSAPI provides no mechanism for the developer to redirect the output of the synthesizer from the speakers to a file or to an output stream. JSAPI is a prime example of emerging standards that are still incomplete and often poorly implemented by third-party vendors.

The Talker class included in the "Resources" section demonstrates a quick-and-dirty workaround for JSAPI limitations. Specifically, it demonstrates how to wrap the speech synthesis function of a JSAPI-compliant engine for easier access. The talker was developed and tested for IBM ViaVoice Millennium Edition and will require IBM's ViaVoice SDK Java Technology Edition, available to developers on the Web at www-4.ibm.com/software/speech/dev/sdk_java.html.

Warning: The Talker workaround is provided for informational purposes only. Consider it experimental and only a temporary patch for the current limitations of the JSAPI.

Talker attempts to bypass the limitations of the JSAPI by implementing an audio loopback function that will capture synthetic speech through the audio input port as the synthesizer is playing the audio (we warned you that this was for experimenters only). You"ll need a full duplex audio card capable of playing and recording audio simultaneously. You'll also need an audio patch cable appropriate for your sound card. Some sound cards will require an attenuating cable between the audio output port and the microphone or audio input port. The safest way to loop the audio output back into the audio input may be to route the audio through an amplifier or mixer with an appropriate audio line out or microphone out port. You'll also have to adjust volume levels manually.

This workaround is viable only because of the ViaVoice speech synthesizer's ability to queue text-to-speech requests, operate asynchronously, and accurately report the start and conclusion of audio output. Talker is able to wrap the synthesizer and treat it as a shared resource.

VoiceXML

VXML is emerging as the standard for scripting phone/voice service applications. Based on XML and supported by a consortium of telephony vendors including IBM, Nuance, and Lucent Technologies, VXML looks promising as a cross-platform scripting system.

If your voice service can be implemented as a relatively straightforward dialogue between caller and server, and if it requires relatively simple error handling and little or no access to native platform capabilities or local resources, then VXML may be the most cost-effective development approach. If your service requires sophisticated or special speech processing capabilities, you might consider a hybrid approach using VXML and a third-party voice component framework like Nuance's SpeechObjects and Voice Channel technologies.

VXML is a young standard, and portable implementations may be expensive and complicated. You can get started with VXML by visiting TellMe Networks. Nuance also provides a lot of helpful information for developers.

Relevant Packages and Standards for Java Developers

The Java Telephony API and the Java Speech API promise to simplify the development of telephone/voice systems and applications…eventually.

JTAPI

JTAPI must be implemented by telephony device vendors and will offer systems developers a comprehensive software platform for telephone devices, call centers, and PBX/CBX systems. Many of its functions go beyond what is necessary for voice/phone application developers.

JSAPI

Application developers can look forward to a unified interface to commercial speech synthesis and recognition engines, but significant oversights in the API render it practically useless for service developers. Notably, JSAPI assumes that all speech synthesis will be delivered through the default audio output device of the user's PC and that all audio input will be delivered through the user's microphone. Audio systems controls for redirecting output to a stream or file are lacking as are controls for specifying an audio input stream. Service developers must look to third-party Java interfaces for comprehensive packages.

VoiceXML

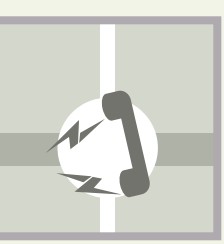
Developers will find a lot of useful VXML information on the Web from IBM, Nuance, and TellMe Networks. Visit the www.vxml.org Web site for a complete specification.

Building Automation Java API (JSR-000060)

Environmental control of office buildings is emerging as a ripe opportunity for cross-platform software. Commercial real estate management companies will automate their control and management functions with standards that work across a broad field of properties.

Open Services Gateway Specification

The home of the future has been a favorite subject of World's Fair prognosticators for decades. The OSG consortium (www.osgi.org) may actually get it right. When broadband opens the big double-wide doorway into homes and offices, the possibility for services will emerge and the demand for applications will follow. Java developers will be influential in



specifying how these services are developed and delivered. If you want to turn on the lights and set the thermostat from your cell phone, watch this specification for the nuts and bolts.

Resources

- ***AnnouncePhonelet.java:*** Demonstrates how to process caller ID packets. Announces an incoming call and the identity of the caller if the phone number (as delivered by caller ID) is contained in a contacts database.
- ***MessagePhonelet.java:*** Source code demonstrating how to record and play voice messages, how to generate synthetic speech from text, and how to use touchtones to control program flow.
- ***Talker.java:*** Source code for a wrapper to simplify use of a JSAPI speech synthesis engine. Has been tested with IBM's Via Voice and Speech for Java SDK.
- ***CallerID.java:*** Source code for a caller ID object that parses unformatted caller ID packets.
- ***PhoneServerLite:*** A multithreaded phonelet host with loadable device handlers and phonelets. Includes javadoc documentation for phonelet framework.
- ***SpeedSerialWin32.dll:*** Win 95/98/NT native library for high-speed serial access. JavaSoft's javax.comm package provides adequate support for serial data transfers at low speeds, but fails at the relatively high speeds necessary to record and play digitized voice with an external voice-enabled modem. The javax.comm package is unnecessarily complicated by an attempt to wrap the parallel and serial ports into a single package. SpeedSerialWin32 simplifies the programmer's view of the serial port and provides reliable high-speed serial transfers.
- ***CommonVoiceModemCommands.txt***

Recommended Reading

- Lindley, C.A. *Digital Audio with Java*. Prentice Hall.
- McClellan, J.H., et al. *DSP First: A Multimedia Approach*. Prentice Hall.
- Pierce, J.R., and Noll, A.M. *Signals*. Scientific American Library.
- Rorabaugh, C.B. *DSP Primer*. McGraw-Hill.

Web Links

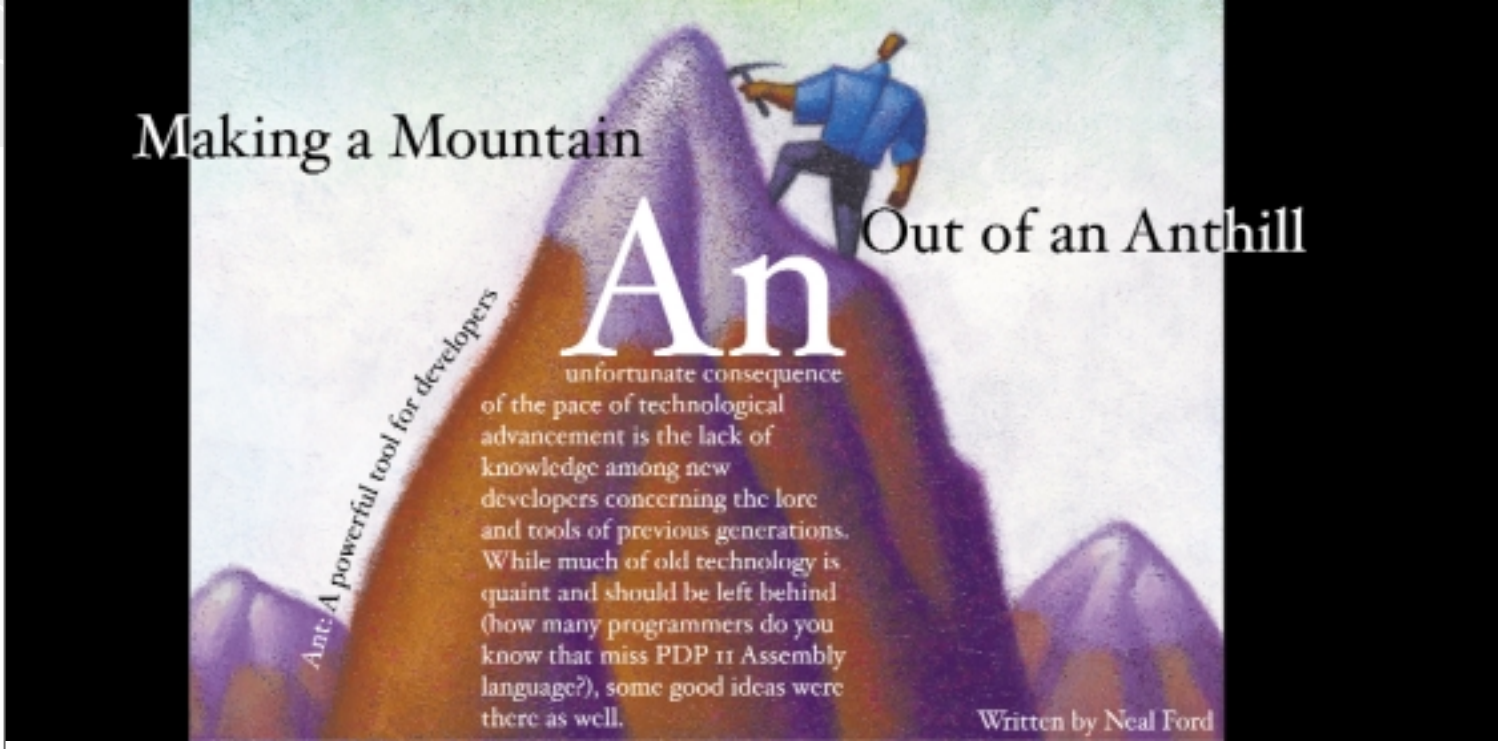
- *A brief history of TouchTone:* www.att.com/technology/history/chronolog/64touch.htm
- *VoiceXML forum:* www.vxml.org/
- *IBM speech technologies:* www-4.ibm.com/software/speech/
- *Caller ID FAQ:* www.ainslie.org.uk/callerid.htm
- *JTAPI:* www.javasoft.com/products/jtapi/
- *JSAPI:* www.javasoft.com/products/java-media/speech/,

Author Bios

Kent V. Klinner III, chief technology officer at TransPhonic, Inc., develops platforms and components for portable and wireless devices. An electrical engineer who still likes to get close to the hardware, he's been developing Java since 1995.

Dale B. Walker is principal engineer at TransPhonic, where she develops applications for mobile and wireless devices. Dale is an electrical engineer with 20 years of broad experience.

kklinner@transphonic.com / dwalker@transphonic.com



One of the not-so-old and not-completely-forgotten programmer utilities of the past is the *make utility*. I'm not suggesting that make isn't still used (especially in the UNIX world), but the percentage of programmers who know what a make utility is and what it does is pretty small. Yet the need for its functionality has never been greater, particularly for large-scale Java projects. I hope this article will return it to the forefront where it belongs.

First, for the unenlightened, a little background on what a make utility does. Then I'll discuss the best thing to happen to Java since cream and sugar – Ant.

Make Utilities

Back in the dark ages of software development (about 10 years ago), most code was written in a good editor (back when the editor *was* the IDE). To make sure it was all compiled and linked correctly, a make utility was used. A make utility is a command line utility that uses a text file that contains a bunch of rules to transform one thing into another thing. For example, the make file would define the relationship between a ".c" source file and the ".OBJ" file generated by the compiler. The make file would then follow the rules (i.e., how to invoke the compiler) to make the transformation happen. You could also define dependencies to control the order in which artifacts were created. For example, you must compile the source files before you deploy them. Make files don't flow from the top of the file to the bottom. You tell the make file what the ultimate target is and it executes the necessary steps (in the correct order) to fulfill the target.

An example is in order. Listing 1 shows a simple make file for a C++ project (all code listings can be found at JavaDevelopersJournal.com). The top of the make file defines what files make up the project. Next, compiler flags are listed as properties of the make file, and the compiler executables are defined as properties. In the last part of the make file, each line defines a transform for how to convert one file into another file. Thus, to convert a .cpp file to an .obj file, you use the compiler command (with the flags) listed in the make file. You'll also notice that the dependencies are implied. In other words, you can't make the .EXE file until the .OBJ files have been created.

Even though this is a simple concept, make files can perform a great service. They exist to handle repetitive, multistep processes that change in content (source files) but not in tasks (how to compile, deploy, etc.). Make utilities exist for all different platforms. In fact, most versions of UNIX have a make utility built into the operating system.

Traditional make utilities are a little ornery, however. The make file syntax must be in an exact format, and the format is specific to the version of the make utility. Different implementations of make utilities require slightly different syntax. As you can see, the syntax of the make file is somewhat daunting – it's like learning an entirely new development language. The make file also defines special wildcards (like the "@&&!" wildcard in Listing 1).

The bigger concern is that make utilities aren't well optimized for Java. For example, Java has very special requirements for package and directory structures. Make utilities don't natively understand this relationship. This isn't to say they haven't been used quite successfully for Java development. However, what was really needed was a new make utility that understands Java. That's where Ant enters the picture.

Ant

Ant, a make utility written in Java for Java, is part of the impressive Jakarta project at Apache. In fact, it's one of the Jakarta subprojects. The Apache developers needed to be able to automate the build process and they couldn't do it gracefully with traditional make files, so they wrote their own.

JRockit

www.jrockit.com

It turned out to be so useful that they released it as its own subproject. And because it's from Apache, it's open source – the source code is freely available. You can find Ant in both binary and source form at http://jakarta.apache.org/ant. To install ant, you unzip it into some directory structure and take the following three steps:

1. Set the ANT_HOME environment variable to point to the installation directory.
2. Add the ant/bin directory to your system path (Ant is started with a bat file in the /ant/bin directory, so you'll probably want to put that directory on your path for easy access).
3. Set the JAVA_HOME environment variable to point to your JDK installation directory.

Ant differs in several ways from traditional make utilities. First, the make file itself isn't in a proprietary file format but is an XML document called *build.xml* by default. Within this document you specify a series of targets. Each target encapsulates some task or set of tasks that you need to perform. Ant also defines a set of built-in tasks to handle everything from copying files and directories to e-mailing someone. For a short partial list of the built-in tasks see Table 1. A full list of the tasks is included with the docs that come with Ant (as of the current version, there are 47 predefined tasks). Moreover, because it's written in Java, you can create your own Ant tasks by subclassing built-in tasks.

ANT TASK	DESCRIPTION
copy	Copies a file, a series of files, or an entire directory structure
cvs	Checks files into or out of the CVS version control package. You can specify the labels and other CVS specific information for the checkout
exec	Executes an external command (i.e., shells out to the operating system to call any command)
javac	Invokes the javac compiler. This task allows you to set the classpath, targets, output location, and a host of other properties
java	Launch another Java application from within the Ant process
javadoc	Invokes the javadoc compiler
mail	Sends an e-mail to a specific address (to notify of build completed successfully, the results of automated unit testing, etc.)
replace	Replaces the contents of one string with another string in a text file
jar	Generates a JAR file
unjar	Unpacks a JAR file
war	Generates a Web Archive format file
sql	Executes an arbitrary set of SQL commands against a database

TABLE | Summary of a few predefined Ant tasks

Generally, you define a build.xml file as a series of targets, each consisting of one or more tasks. Listing 2 shows a simple build file for Ant. Notice that the top of the file starts with an "init" task. This is typical in that it allows you to define properties that can be used throughout the build document. You can think of these as global variables that can be used anywhere within the build document. This is typically where directories, compiler settings, and flags are set up. By defining them at the top of the file, you can make changes to the values that flow throughout the entire document. The next target, named *usage*, allows the developer to invoke the Ant make file and find out what targets are available. This is help for this particular make file. The third target is called *prepare*; the fourth, *prepare-src*.

Automated builds can sometimes take a long time for large projects. It would be convenient if you could continue working on the project while the build is proceeding. Thus one of the early tasks typical in a build file is a task to create a build working directory and copy all the source files into it. The build utility can then work on the source code in the new make directory while you make changes to the mainline code.

Notice how Ant handles directory structures. Because Ant was written for Java, it understands how Java packages and directories are related to one another. Ant defines a special file-based wildcard: "**". This is like the normal wildcard "*", which matches anything in the current directory. However, the "**" wildcard matches everything in this directory and all its subdirectories. So if you tell Ant to copy /src/**/*.java, it will copy all the Java source files in all the subdirectories of the /src directory. In addition, when Ant copies source files, it leaves the directory structure in tact.

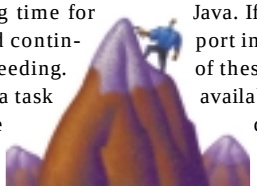
The next target in the build file is the Java compiler invocation. It will compile all the source files and place them in the destination directory defined by a property above. This task also shows how you can specify classpath information for tasks that need a classpath. You can define the classpath attribute in two ways. You can use a semicolon-delimited list, much like the classpath environment variable is defined. But that creates an attribute that's hard to read. The alternative, shown here, is to use a nested property. In this case the classpath is defined as a special attribute, which has its own attributes, making it much easier to see what's on the classpath and make changes.

A Javadoc task is defined next. You probably don't want the Javadoc process to run every time you build your project. You'd probably rather run Javadoc only at certain stages in the project. To invoke Ant for a particular target, you can pass the name of the target on the command line. Ant will then try to fulfill that target. However, each target may have a "depend" attribute. Ant will ensure that every dependent target is run successfully before executing the next target. Notice that the "compile" target lists "prepare-src" as the target it's dependent on. So, like traditional make files, the flow of execution doesn't necessarily go from the top of the document to the bottom. The dependencies between the targets determine the order of execution. You can also define a default target at the top of the file – this is the target that will be fulfilled if no specific target is passed on the command line. Thus the Javadoc target will be executed only when you specify that you want Javadoc.

The last target is the "clean" target. It goes through and cleans up all the working directories created by Ant.

When you run Ant, you can pass it an XML document and a target. If you've named your build file "build.xml" and it's in the directory where you start Ant, you don't have to specify the build file name. Moreover, if you're building the default target, you don't have to specify a target either. Most of the time, therefore, you'll just invoke Ant on the command line. Ant will show each target that it executes in turn, printing status messages for individual tasks. Figure 1 is an example of a typical successful build. However, if any task can't complete successfully, the entire build process is aborted. Figure 2 shows an example of a build that failed.

Because Ant is written in Java, it can be extended using Java. If you have a particular tool that you need to support in Ant, you can build an Ant task to handle it. Some of these custom tasks have already been written and are available from the Ant Web site. For example, many developers use Visual SourceSafe as their version



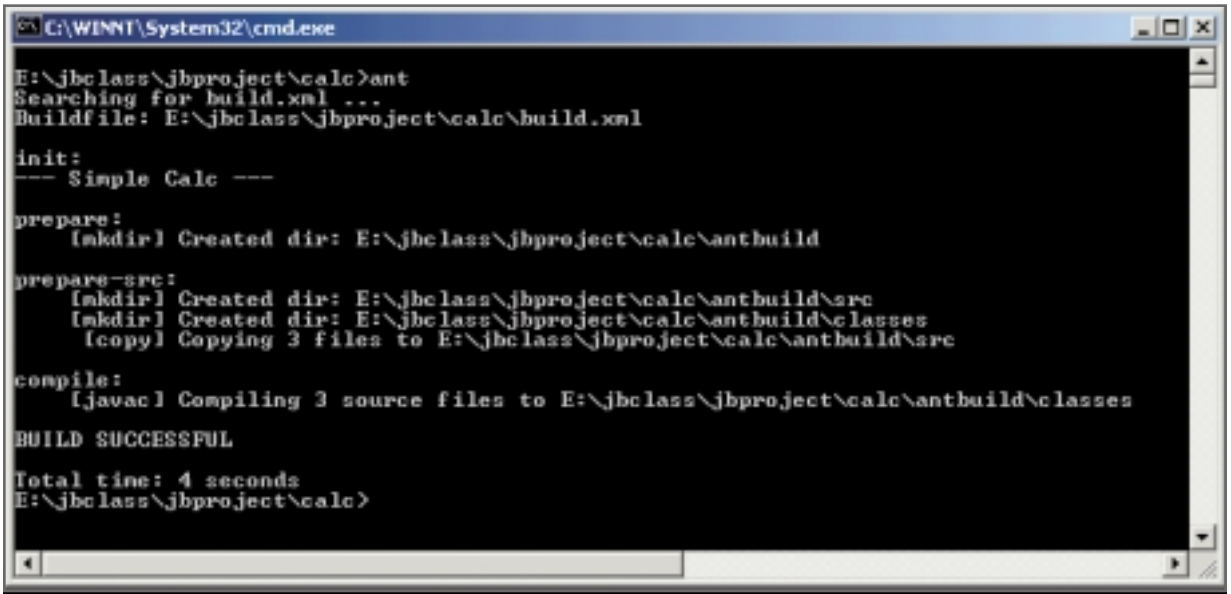


FIGURE 1 Ant shows the status of each task and whether or not the build succeeded.

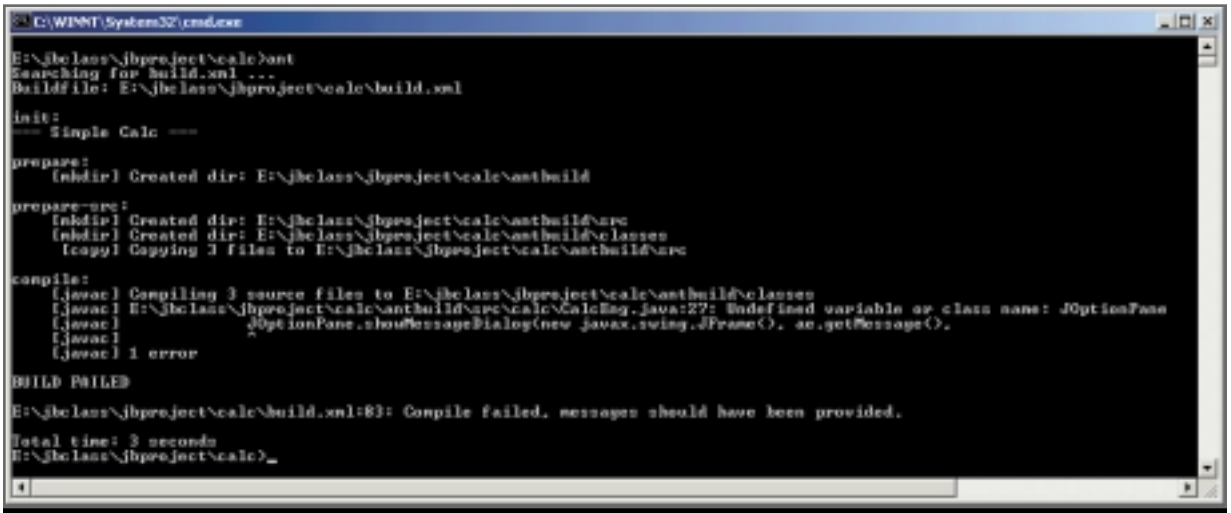


FIGURE 2 When a build fails, Ant stops on the task that fails.

control package. Ant has support for CVS built in, but there's no support for VSS. However, if you download the optional.jar file from the Ant site, you'll discover that someone has already written the integration for VSS for Ant. The optional JAR file contains several tasks that were written and contributed to the Ant project that aren't mainstream enough to become built-in tasks but are still useful. To use one of the optional tasks (or one of your custom tasks), you have to make Ant aware of what class to use to define the task. This can be done with a taskdef task, generally placed in the init section of the build file. A sample task registration is shown below:

```
<taskdef name="mytask"
         classname="com.mydomain.MyVeryOwnTask"/>
```

You can use the MyVeryOwnTask as if it were a built-in task. If you want to register a new task with Ant more permanently, you can add the task to the default.properties file in the org.apache.tools.ant.taskdefs package. To use the optional VSS task, add a taskdef line to your build file and use the following syntax to invoke it:

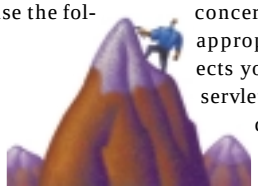
```
<vssget localPath="C:\mysrc\myproject"
        recursive="true"
        label="Release1"
        login="me,mypassword"
        vsspath="/source/aProject"
        writable="true"/>
```

Details on both the optional tasks and how to build your own tasks are included in the Ant documentation.

Enterprise Deployment

As you can see, Ant makes it very easy to consolidate all project construction information in a single location. You can define classpaths and other build artifacts once and reuse them throughout the build file. For simple projects like the one shown above, however, most modern IDEs handle some of these chores (although none that I have seen handle it as well as Ant). Ant really begins to shine when you look at more complex applications that have more complex dependencies or complicated distribution requirements.

A major headache in building Web applications in Java concerns the deployment of the required files to all their appropriate locations. For example, in most Web projects you have a mixture of HTML, JSPs, JavaBeans, and servlets. All these files (except HTML and JSP) go into different destinations. Each Web server has its own



> Ant's ability to guarantee consistent builds and correct deployment, and its automatic unit testing **make for a powerful tool indeed**

definitions of where these files belong. Although this has gotten better since Sun introduced the concept of a WAR file and the ensuing directory structure, the deployment headache still exists. If you forget to copy even one of these files when you deploy your project, the magic just doesn't work. This sort of tedious task is exactly why tools were built. Obviously, you can build an Ant build file that can handle not only the compilation chores for each Web project, but these annoying deployment chores as well.

One of the nice characteristics of Ant is that it does the minimum amount of work necessary. So, if a file already exists in the target location and it has the same timestamp, Ant won't bother copying it. This is contrary to the solution that many developers currently use, the lowly batch or script file. The disadvantage of batch and script files is that they generally do the most amount of work necessary rather than the least. Some scripting languages allow you to build in more intelligence, but you must remember to add it. Ant does the smart thing automatically.

If you look at the build and deployment process for multiple Web projects, it looks as though they do basically the same things, just for different sources and destinations. Rather than build a separate Ant build file for every Web project, it seems as though you could build one Ant build file and change just a few characteristics to make it work for almost all Web projects. This generic build file for Web projects (see Listing 3) makes a couple of assumptions about the way you have your project defined. First, it assumes that there is a source directory (named /src, but this detail is easy to change) that consists of a directory/package structure containing your Java source files. It also assumes that the JSPs and HTML are defined in the /src directory without being inside the package hierarchy. Last, it assumes that the JavaBeans used as library classes by your servlets are not in the same directory or directories as your servlets. You may want to alter the build.xml file to change some of these assumptions.

A nice Ant feature demonstrated in Listing 3 is the ability to define Ant properties outside the Ant build file and import them. Listing 4 shows the properties for a specific Web project. Ant uses standard java.util.Properties files and a special property attribute to import properties. To import the properties file in Listing 4, the following task line is defined in the init target:

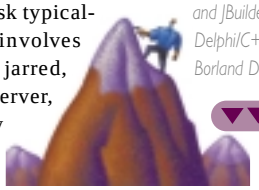
```
<property file="project.properties"/>
```

This imports the properties in the named file, which become indistinguishable from properties defined directly in the Ant build file. This mechanism makes it easy to define generic build files that can be customized without even editing the build.xml file.

The other monstrous build and deployment task typically found in enterprise Java development involves Enterprise JavaBeans. They must be developed, jarred, handed to a deployer tool for the application server, copied to a deployment location, and optionally unjarred in the appropriate place so the client has access to stub code. This is obviously a job for Ant! In fact, there is a relatively new set of Ant tasks defined especially for EJB tasks. Currently, the tasks defined for EJBs support only WebLogic 4.5.1 and 5.1, although I'm sure other tasks will appear as needed. If you don't use WebLogic, here's your chance to add to the open source movement! Of course, as long as your application server supports command line tools, you can use the exec task in Ant to invoke the tools for your application server. In fact, before there were specific tasks for EJBs, I wrote an Ant task to execute WebLogic's ejbc compiler to generate all the necessary code for deployment. Just like WebLogic, most vendors give you Java versions of their deployment tools, all of which are easy to launch from within Ant. One advantage this has over batch files is that Ant can spawn the VM in a new thread rather than in a new process. Thus the start-up overhead is much less when Ant launches a Java application versus when a batch file launches the same application. In an organization that used batch files to check out, build, and deploy a large number of EJBs, I wrote an Ant build file that reduced the amount of time required from 47 minutes to 15 minutes! The entire time savings were realized by doing the minimum amount of work rather than the maximum and by spawning threads instead of processes.

The last supported task type in Ant that I'll mention is integration with the open-source testing framework JUnit. In a nutshell, JUnit defines a framework to automate unit testing for your Java classes. It includes both a command line and a graphical test environment. The JUnit task is defined in the optional JAR file and can invoke JUnit tests as part of the build process. The typical scenario involves building classes and defining tests, and making sure they're run each time the project is built. If subtle bugs appear during subsequent development, the unit test fails and can optionally fail the entire build process. Alternatively, you can define a task to e-mail the build manager whenever a class fails its regression test.

Ant's ability to guarantee consistent builds and correct deployment, and its automatic unit testing make for a powerful tool indeed. It automates part of the development process that should be automatic. Developers are too busy with more important jobs than making sure that files are copied correctly. Ant is powerful, extensible, convenient. And the source is included and the price is right!



Author Bio

Neal Ford is vice president of technology, DSW Group, Atlanta, Georgia. He is also the designer and developer of applications, instructional materials, magazine articles, and video presentations, and author of the books Developing with Delphi: Object-Oriented Techniques and JBuilder 3 Unleashed. Neal has been the featured speaker for the Borland Delphi/C++Builder/JBuilder World Tours, and has spoken extensively at annual Borland Developers Conferences worldwide.

nford@thedswgroup.com

Host Centric

www.hostcentric.com/jdj

HP WebGain

www.webgain.com/javaone_hp.html

JNI Programming in C/C++

Additional challenges of using the IBM e(logo)Server iSeries

WRITTEN BY
BLAIR WYMAN

If you're familiar with the Java Native Interface (JNI), as this article presumes, you know that it's tailored primarily for C and C++ programmers. Compile-time support for JNI in these languages comes straight from the Sun specification, and is frankly a work of art.

The architects of the JNI had a terrifying three-part task: to tame the hydra of platform-specific issues inherent in so-called "native" code, provide a way to use native code in Java, and to do so in as "portable" a fashion as possible. The ubiquity and standardization of C and C++ made them the natural choices for preferred native languages, and their affinity to Java is apparent to anyone who has programmed to the JNI.

If you're familiar with IBM's e(logo)Server iSeries machine, you know it supports a wide range of programming languages, including:

- C (in several incarnations)
- C++ (quite recently)
- RPG (Report Program Generator)
- COBOL
- CL (Command Language, the iSeries' workhorse)
- qsh (POSIX-based scripting language)
- MI (Machine Interface assembler language)
- Perl

In contrast to most UNIX-based machines, however, the C language has never been at the foundation of the iSeries machine's operating system; in fact, C is a relative newcomer to the platform at the applications level. The internal development language has varied over the years, but if any language deserves the foundation role on iSeries, from an applications-development standpoint, it's Report Program Generator (RPG).

The heritage of today's iSeries machine dates back quite far, by modern standards. You probably recognize the most recent progenitor of the iSeries platform, the IBM AS/400 System announced in 1988, but it's actually the IBM System/38 that deserves the credit for being the seminal implementation of this truly unique computer architecture.

The unique features of the iSeries architecture are too numerous to recount here, but a couple of salient points will motivate the discussion that follows.

The original architects of the iSeries chose not to differentiate between "working storage" and "disk storage"; it's logically just one big pool of something colloquially known as *Single Level Store* (SLS). One advantage of this scheme in a database-centric system is clear: process-specific data movement is all but eliminated. Data can be shared among processes, without the overhead of ensuring that process-specific virtual address images of the data are seen in a consistent form.

Obviously, to accomodate for future growth, this SLS scheme requires a very large address space since everything on the machine is in one big "addressable universe." To satisfy this requirement, the designers of the System/38 specified a 16-byte (that's *byte*, not *bit*) pointer. Well, that was over 20 years ago, and the 16-byte pointer has so far stood the test of time.

Sharing a single address space across the entire machine? How can there be any hope of security between processes and users? The answer is in the notion of the pointer being "tagged." If a pointer operation is performed by the machine, then the pointer is tagged (i.e., made valid). If any sort of operation, other than a pointer operation, changes the storage, the tag is cleared and the pointer is invalidated.

As you can imagine, pointers of this size are...um..."unusual," even in today's diverse mix of systems. It's the very size of our 16-byte pointer that's probably the most cantankerous issue when porting Java native method code to the iSeries machine, as we'll see.

So, how could this unusual memory architecture even hope to support something as modern as Java? Especially, how could it support Java well enough to allow the iSeries (then AS/400) JVM to capture the top spot in four separate industry-standard Java benchmarks last summer? One part of the answer is simple, yet profound: Java doesn't know about "pointers" and "addresses." Hallelujah. With the introduction and increasing popularity of Java, the iSeries is on level ground from a programming standpoint.

However, as soon as we start talking about JNI and native code, we're talking mainly about C and C++; pointers and addresses are back in the spotlight. And,

Corda

www.corda.com

> JNI programming on the IBM iSeries system poses some extra challenges to the programmer, but none that are insurmountable

of course, there are other wrinkles. The iSeries is basically an EBCDIC machine, but Java and the JNI prefer UCS-2 or UTF-8 character encodings. Most machines run the JVM at a sort of "application-level," while the iSeries effectively embeds the JVM into the operating system.

There are five basic issues to consider when porting native method code to the iSeries machine:

1. JNI reference types (e.g., jobject, jclass, jstring, etc.) are implemented as 32-bit signed integers, instead of pointers.
2. In C and C++ source code, literal strings and character values are encoded in EBCDIC, by default.
3. The JNI jlong data type is implemented on the iSeries using a C struct of two integers, instead of a true 8-byte integer.
4. The simple pointer type in C and C++ is 128 bits, as discussed earlier.
5. Direct addressability into the garbage-collected heap is never permitted.

Let's examine each of these in turn:

1. *Object references are integers*

This is a nice place to begin, since it's only a minor factor in the porting effort, but brings up some fundamental iSeries issues right away.

The iSeries pointer in C and C++ is a different sort of animal, as we saw earlier, and this functionality comes with a price: manipulating pointers is inherently expensive on the iSeries.

When the JNI specification came out, all the object references were implemented by Sun as pointer types. However, Sun's object reference pointers are "opaque" – there was no "layout" defined at the end. Since these pointers are truly just handles, we asked Sun for permission to implement our object references as integers instead of pointers.

The main impact of this change occurs only when compiling C++ native method code. The JNI specification for C++ uses a class hierarchy to allow some type-conformance checking to occur at compile time. For instance, in C++ you would never be allowed to assign a <code>jString</code> object into a <code>jBooleanArray</code> variable without an error message. When the references are integers, this type-conformance checking is lost. A minor impact is that the null reference becomes the integer zero, instead of a null pointer value.

2. *Codepage considerations*

The iSeries is an EBCDIC machine, and Java's Unicode heritage is ASCII-based, so there's a sort of conflict between Java's native side and the iSeries. This conflict is mostly avoided when you stick to Java, but once you start writing native method code, Java's wonderful platform-neutrality is necessarily reduced.

The JNI expects all character data to be provided either in Unicode (the <code>jchar</code> type), or in the modified UTF-8 representation of Unicode (where the NUL character is represented using the two-byte encoding of 0xc080).

UTF-8 has the interesting property of being, itself, composed partly of the 128 characters of the ISO 8859-1 Latin character codepage. That is, all ASCII character arrays are already UTF-8 – a quality that simplifies coding native methods on ASCII platforms.

This issue is really twofold, depending on whether the character data is literal or dynamic.

Literal Strings

Using the ILE C and C++ compilers for the iSeries system, the default encoding for literal character data is EBCDIC. However, this compiler default can be overridden using a #pragma statement in the source code.

```
#include <stdio.h>
#include <jni.h>
/* literal strings here are still
EBCDIC */
char *fmt = "%d:%s\n";  /* EBCDIC
*/
char *message = "Return code non-
zero!\n"; /* EBCDIC */
#pragma convert(819)
/* starting now, all literal
strings are ASCII (819) */
/* NOTE: ASCII is NOT exactly the
same as UTF-8!     */
void main(int argc, char **argv) {
  int i, rc;
  for (i=0;i<argc;++i) {
printf(fmt,i,args[i]); }
  rc = asciiFunction("Great googly-
moogly!"); /* ASCII */
  if (rc) printf("%s", message);
/* OOPS!!!  Error! */
}
/* Formats MUST be */

/* in EBCDIC ONLY  */
#pragma convert(0)
char *backInEbcdic = "I am EBCDIC,
hear me roar";
```

Since JNI requires UTF-8 input, it would naturally be wonderful if we could just specify some value *nnn* in our #pragma convert statements that would magically transmogrify the literals into a UTF-8 encoding. Unfortunately, although the iSeries supports UTF-8 for dynamic string conversions (as we'll see in a moment), it does not yet support the use of UTF-8 in the #pragma convert statement.

Instead, a value of *nnn* that most closely fits the bill is codepage 819, which is ISO 8859-1 Latin. The low-order 128 characters of this codepage are identically UTF-8.

We have to be careful, though, to make sure that all literal strings contain only invariant ASCII – any use of an 8-bit character whose most-significant bit is set requires that the character be represented in its two-byte UTF-8 form. Fortunately, most uses of ASCII for JNI parameters such as class and method names are already likely to be UTF-8. Literals containing character values greater than decimal 128 can be represented using escapes in the source string, as this example shows:

```
  /* \u00EB is 'ë' -- the ISO
8859-1 Latin 1 code for (e-diare-
sis)
   * The UTF-8 value for this
character is the two-byte sequence
0xC3AB
   */
  p = "Zo\xc3\xab";  /* UTF-8 for
Zoë */
```

Dynamic Strings

In contrast to literal strings, the value of dynamic strings is not known until runtime. These strings may become parameters to the JNI, and therefore

Compoze

www.compoze.com

might have to be converted to "modified UTF-8." The standard API for the purpose of codepage conversion on the iSeries is <code>iconv()</code>.

The iconv() API accepts two codepage numbers – one for the "from" codepage, and one for the "to" – and supports both UCS-2 and UTF-8 codepages. However, be forewarned. In the (admittedly unusual) case where your C string might contain an embedded NUL character, the UTF-8 conversion of iconv() preserves it. Remember, JNI takes "modified UTF-8," where the NUL is a two-byte character.

3. *The jlong data type*

The <code>jlong</code> data type is the JNI equivalent of the Java <code>long</code>, an 8-byte integer value. At least it's supposed to be.

When the JNI was first being implemented on the AS/400 system, the C language did not have an 8-byte integer value (although COBOL did, ironically). So, the only alternative for our implementation was to define the <code>jlong</code> as a structure, with two 4-byte integers.

Now, when the 8-byte integer came to the C compiler on the AS/400, we had already shipped a release of JNI using the structure form of <code>jlong</code>, and we realized that if we changed our support to use the 8-byte integer form, we would be sacrificing forward-compatibility in existing native method libraries. The iSeries team is loathe to introduce incompatibilities that break existing code. Therefore, despite the fact that the ILE C and C++ compilers have supported 8-byte integers for a couple of releases, the JNI is stuck with the struct form of jlong.

The impact of this fact on the porting effort can be widespread but generally straightforward. For example, consider this C native method code:

```
JNIEXPORT jlong JNICALL
Java_TestJava_getlong
(JNIEnv *env, jobject obj)
{
  long y;
  somefunc(&y);  /* pass y by ref-
erence to some routine   */
  return y;       /* Uh-oh! y is
not compatible with jlong */
}
```

The "fixed" version of this code looks like:

```
JNIEXPORT jlong JNICALL
Java_TestJava_getlong
(JNIEnv *env, jobject obj)
{
  jlong x;        /* here
is our compatible return
value */
  long y;
  somefunc(&y);  /* pass y
by reference to some rou-
tine   */
  JavaI2L(y,x);  /* con-
venience macro from
jni_md.h        */
                 /* sign-
extends if necessary
*/
  return x;       /* OK */
}
```

4. *"Pointer hiding" in Java objects*

One of the most difficult issues with porting native methods to the iSeries is the issue of "pointer hiding." An all-too-common idiom in native methods is to take the address of some native storage and to store that address in a <code>int</code> or <code>long</code> field of a Java object using JNI.

```
someObj *somePointer;
fid = (*env)-
>GetFieldID(env,clazz,"intField","I
");
(*env)-
>SetStaticIntField(obj,clazz,fid,(i
nt)somePointer);
```

As we discovered earlier, the iSeries pointer is 16 bytes in length, so it simply doesn't fit. And even if it did, the modification of the pointer as an integer would clear the tag, rendering the pointer invalid. So we see there's a problem.

There are two types of solutions – one that's robust and expensive, and one that's lighter in weight yet more complicated to get right.

The robust, heavyweight solution relies on two key facts:

1. The iSeries defines a special class, com.ibm.as400.system.MiPtr, which can be used to hold 16-byte pointers,
2. Object references on the iSeries are integers.

The scheme is simple. Each time the native code stores a pointer into a Java int or long field, change the code to create a com.ibm.as400.system.MiPtr object, store the pointer into the object, take a global reference to the object, and store that reference into the int or long field.

In order to use this scheme, you have to bind your native method library code with the QJVAJNI service program, which exports two functions called "GetPointer" and "SetPointer". Here are the before and after pictures:

- ***Before:***

```
void     *ptr;     /* pointer we
want to store in the int field
*/
jclass   otherClz; /* class con-
taining static int field
*/
jfieldID otherFid; /* fieldID of
the static int field in otherClz */
/* DO NOT try the following at home
*/
(*env)-
>SetStaticIntField(env,otherClz,oth
erFid,(int)ptr); /* UGH */
```

- ***After:***

```
void *ptr;          /* pointer we
want to store in the int field
*/
jclass otherClz;   /* class con-
taining static int field
*/
jfieldID otherFid; /* fieldID of
the static int field in otherClz */
jclass pClz;       /* class ref to
com.ibm.as400.system.MiPtr     */
jobject pObj;      /* object ref
```

```
to instance of MiPtr we create   */
jmethodID pCtor;   /* method ID of
the MiPtr constructor  <init>  */
jobject gRef;      /* global ref-
erence to new instance  (why?)
*/

/* create instance of
com.ibm.as400.system.MiPtr */
pClz =(*env)-
>FindClass(env,"com/ibm/as400/sys-
tem/MiPtr");
pCtor=(*env)-
>GetMethodID(env,pClz,"<init>","()V
");
pObj =(*env)-
>NewObject(env,pClz,pCtor);
SetPointer(pObj,&ptr);
/* NOTE: pass ADDRESS of ptr */
gRef =(*env)-
>NewGlobalRef(env,pObj);
(*env)-
>SetStaticIntField(env,otherClz,oth
erFid,(int)gRef); /* OK */
```

The lightweight solution is more complicated to get right. Basically, you just allocate an array of pointers in C static storage, store the pointer into the array, and then store the integer array index into the Java int or long field.

However, now we have the issue of possibly synchronizing many threads, since the allocation of a "slot" in this table of pointers must be performed atomically. Also, how do we know when the pointer should be cleared from the table? A deep understanding of the application will be required to get this right.

5. *Direct addressability to objects in the garbage-collected heap*

One aspect of the iSeries JVM implementation is its integration with the underlying platform. This integration is tight enough to allow Java-to-Java calls to be made very quickly using a "machine addressing" scheme that avoids the overhead of 16-byte pointers. However, as soon as native method code comes into the picture, so do the 16-byte pointers.

Selected JNI interfaces are designed to provide addressability to the elements of a Java array, or to the characters of a Java String object. For instance, the <code>Get<PrimitiveType>ArrayElements</code> or <code>GetStringChars</code> interfaces return an address to the caller: an address that may or may not be directly addressing the actual object.

On most platforms, these addresses actually point directly into the Java object in the heap. On the iSeries system, however, the JNI programmer is never provided with such direct addressability – these JNI interfaces always return the address of a copy of the data. Even the so-called "critical" interfaces added to the JNI in Java 2 – GetPrimitiveArrayCritical and GetStringCritical – return the address of a copy of the data.

This decision not to expose the garbage-collected heap to any user-level addressing in the iSeries JVM was not made lightly, but it is incontrovertible. If a JNI programmer could get the address of an object in the heap, it would be possible for that code to accidentally walk over storage and corrupt the state of the JVM. This restriction is one clue to the robust nature of the iSeries JVM.

Summary

JNI programming on the IBM iSeries system poses some extra challenges to the programmer, but none that are insurmountable. If the programmer is lucky enough to be dealing with a well-designed body of native method code, code that avoids some of the pitfalls we've discussed, the port to the iSeries is a relative breeze.

Author Bio

Blair Wyman is a software engineer working for IBM in Rochester, Minnesota, home of the IBM iSeries.

blair@blairwyman.com

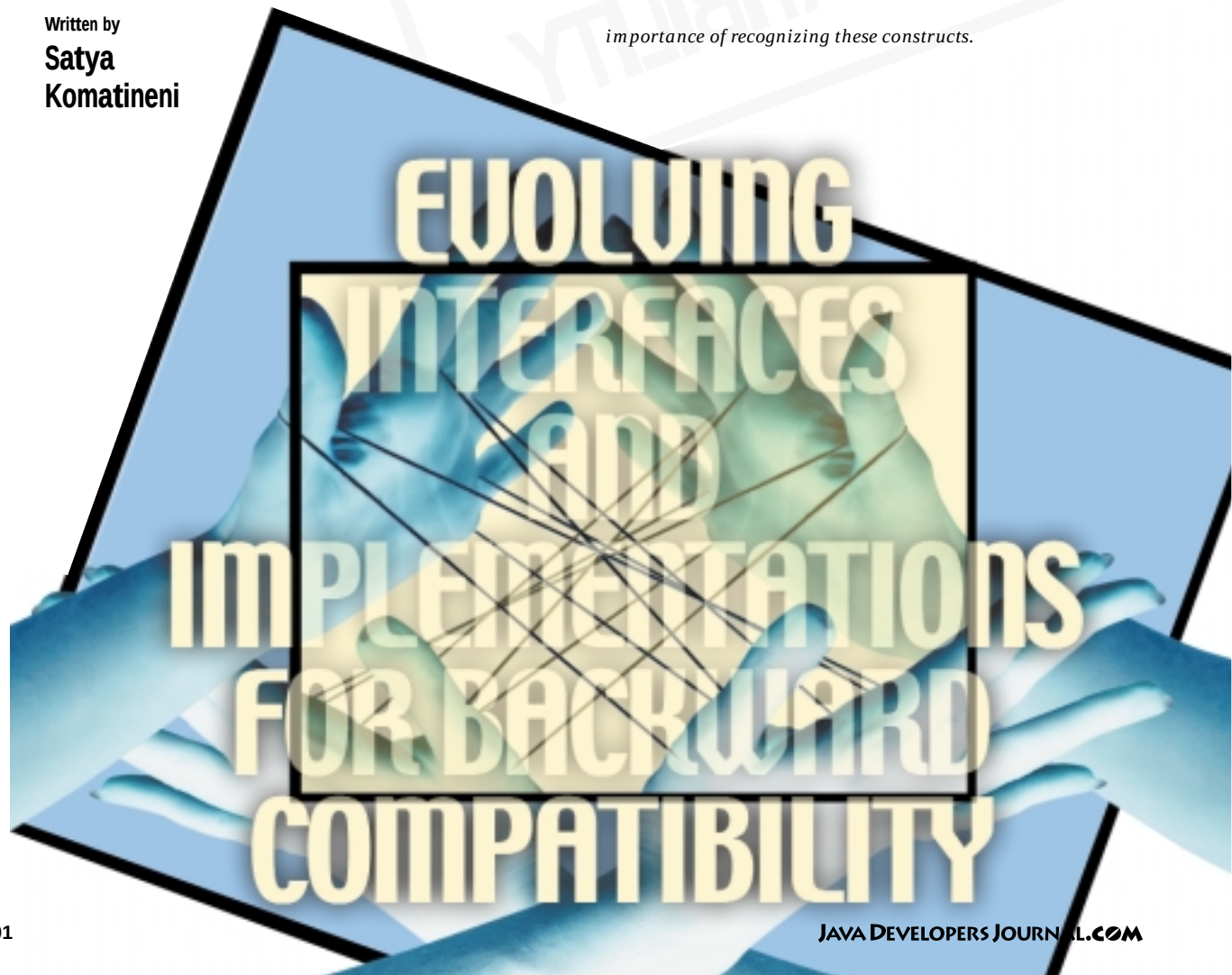
Elixir

www.elixirtech.com

Using simple application-level factory services

C++ brought into vogue the concept of interfaces, abstractness, and implementations. Java went a step further and formalized them with proper keywords for each of the concepts. There are a substantial number of patterns in which interfaces, abstract classes, and classes can be combined for various purposes. You only have to look at the book Design Patterns, *by Erich Gamma et al (Addison-Wesley), to realize the importance of recognizing these constructs.*

Written by
Satya Komatineni



Constructing a software program that you expect to have a long life is an attempt to evolve its functionality over time, without abandoning older behavior for backward compatibility. This is a lot more essential when you're designing libraries and frameworks. The newer libraries should work with older systems that were designed for the older releases of a library. Here are some concrete examples of why this is important and how to accomplish this in situations where every programmer is encouraged to adapt the scheme owing to its simplicity.

What Is Evolving Functionality?

To understand what evolving functionality is, let's walk through the design of a very simple class in Java. Here's the class:

```
class LogFacility
{
 public log(final String message)
 {
  // implementation
 }
}
```

LogFacility provides a mechanism to log messages. Everyone on the team would use this functionality as follows:

```
LogFacility lf = new LogFacility();
lf.log("log this message for me");
```

As it's simple and useful, it's hard to argue against it. Now I take this utility and ship it to two different teams. One team wants to log to a file and the other to stdout. I improvised and came up with StdoutLogFacility and FileLogFacility, and suggested that the programmer use the appropriate logging facility as follows:

```
StdoutLogFacility lf = new
StdoutLogFacility();
lf.log("log this message for me");

FileLogFacility lf = new
FileLogFacility();
lf.log("log this message for me");
```

You'd quickly realize that irrespective of what log facility is constructed, it gets used the same way. This is a cue to design an interface for the usage of the log facility as follows:

```
public interface ILogFacility { public
log(final String message); }
```

StdoutLogFacility and FileLogFacility would implement this interface. In this scenario the programmer would do the following:

```
ILogFacility lf = new
StdoutLogFacility();
lf.log("log this message for me");
ILogFacility lf = new FileLogFacility();
lf.log("log this message for me");
```

As you can see, irrespective of the nature of the construction, an object of type ILogFacility is used the same way.

This brings us to an important observation in OO programming – *construction is different from usage*. In other words, the construction interface of an object is different from its usage contract/interface. We can delegate the construction of the ILogFacility to any number of mechanisms, but the usage remains the same. Using this property let's investigate a method where the programmer doesn't have to do the following:

```
ILogFacility lf = new FileLogFacility();
```

This line of code is hard-coding the construction of an implementation. In doing so we create a compile-time dependency with the implementation. And we can't substitute StdoutLogFacility for FileLogFacility because that dependency is necessitated by the compiling. So, in a general sense, you can't really vary the implementation of a given interface at runtime if we use the "new" option for construction. Also you'll have to know the name of the implementation class, which may not even exist at compile time. What to do? The solution lies in understanding the factory pattern. Here's an example:

```
class ObjectFactory
{
 public static Object create(final String
   objectName, Object args )
  throws CreationException;
}
```

You can attempt the following:

```
ILogFacility lf = ObjectFactory.create
   (ILogFacility.NAME, null);
if.log("log my message"):
```

As a programmer you've delegated the construction of an interface named ILogFacility.-NAME to a factory. This factory could look up a system-wide properties file for a class that's responsible for this interface and load it at runtime. Let's look at one such properties-file entry:

```
Objects.loggingObject.className=com.your_
   company.StdoutLogFacility;
```

Assuming ILogFacility.NAME is "Objects.loggingObject", it is not difficult for the factory to do the following:

```
public static Object create(final String
   objectName, Object args )
{
   Properties systemWidePropertiesFile;
   String className = systemWide
```

When you're specifying target implementation objects via a properties file, it's possible that anything might go wrong in the construction process, resulting in an exception.

```
        PropertiesFile.getValue
        (objectName);
    Object o = Class.forName(className);
    return o;
}
```

Using this mechanism you can improvise StdoutLogFacility gradually without impacting its users. For example, you've improved logging but don't want to replace the older implementation until it's been field tested. You can cut and paste the source code of StdoutLogFacility to another class called StdoutLogFacility1 and replace your properties file as follows:

```
Objects.loggingObject.className=com.your_company.Stdout
   LogFacility1
```

This way the older clients can continue to use the older implementation while you migrate to the new one. You can continue to evolve this functionality while incrementing the numbers. The same facility can provide a radically different implementation if necessary. For example:

```
Objects.loggingObject.className=com.your_company.Socket
   LogFacility
```

logs all your messages to a socket.

This discussion isn't limited to logging; any of your core modules can follow the same pattern, which allows them a path for evolution while maintaining backward compatibility.

To summarize the points so far, we've learned the following:

1. Construction is different from usage
2. Use interfaces, not implementations for, key areas of functionality
3. Use factories to construct implementations, at runtime
4. You can use properties files to associate implementations with their interface names
5. Having an interface.NAME as a standard allows a programmer a simple reference to instantiate an interface

Usage Pattern for an Evolving Implementation

Let's reexamine how we can utilize an interface. The emphasis here is that you get your implementations via a global factory instead of "new"ing them explicitly. This allows for runtime variations of the implementation. For example:

```
MyInterface i = (MyInterface)ObjectFactory.create
   (MyInterface.NAME,null);
   i.useMethod();
```

Let's examine the MyInterface.NAME:

```
interface MyInterface
{
    public static final String NAME =
        "MyInterface";
    .. other public method signatures
}
```

Using a synonym like "interface.NAME" instead of a literal string, we reduce the risk of getting the name wrong. If every interface has the same public synonym, it's just that much easier for the programmer to instantiate an implementation for it.

Interface Definition for an Application-Wide Factory Service

Now we'll discuss the factory object in more detail and define some desirable characteristics. The idea is we'll use this factory object to create all our implementations for the desired interfaces. By that I mean to choose interfaces that are going to be instantiated by factories. Once decided, these interfaces can be instantiated by the same global object factory. The aim of this section is to define a simple interface for such an application-wide factory service.

Static vs Nonstatic Method

The method "create" of the ObjectFactory is declared static and public. It's public for obvious reasons and static so the programmer can invoke it without an object reference. For example:

```
ObjectFactory.create(..)
```

instead of

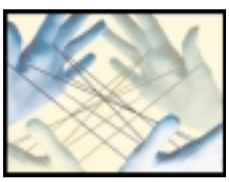
```
ObjectFactory of = new ObjectFactory();
of.create(..)
```

There are some exceptions to this rule for example, if you want to evolve the functionality of the factory itself. The method also throws a suitable exception indicating any failures in the construction process. There are some interesting consequences to the nature of this exception, which I'll cover at the end of this article.

getObject vs createObject

At this point I'd like to change the name of the "create" method of the ObjectFactory to "getObject". create() implies that an object is being created every time the method is called. This may not be true for a number of reasons. Certain objects you may want to create only once, others with each request, and some as part of a pool. Because of this, getObject() suits the semantics of this intention more accurately.

Passing Arguments



Certain objects require construction arguments. How do you pass them? The arguments are passed via the second parameter to the function call. This could be any object, including a collection of objects that get passed

to the constructor of the target object. This can get sophisticated with EJBs when the constructors are invoked through reflection and the correct constructor called. But for simplicity the classes can have default constructors, with a well-known "init()" method that accepts the passed parameter as an argument.

The implication is that programmers not only need to know the interface name of the object that they want to instantiate, but also the nature of its arguments. This I call the *construction interface of the object*. This needs to be published and adhered to by all the implementations. And it's the responsibility of the programmer to pass the correct set of arguments. Otherwise an exception is thrown by the generic factory.

Caching Strategy

Once you start using this facility, it won't be long before you start asking, "How can I limit the number of object occurrences of a specific class?" For example, singleton. To facilitate this a factory can use multiple strategies to cache object instances. The factory can save the instantiated objects in a hashtable keyed by their interface names. So, when a repeated request comes in for an interface, the factory can return the previously constructed object. But how does the factory know to cache an object? You can specify in the properties file that a certain interface needs to be cached, which is how it's done in EJBs at deployment time. Specifying the number of object instances at deployment time is more generic. But for systems of medium complexity, you may want to define an ISingleThreaded interface and implement that interface to force multiple instances of the object. This avoids the errors that might occur due to lack of knowledge at deployment time, or you could provide both facilities.

Name-Based Caching May Not Be Sufficient

Interface name-based caching sounds good until you consider the fact that a given implementation can support multiple interfaces. In this case you'll have two such objects cached when one would have been sufficient and even wrong at times. I believe the right approach is to use caching based on class names instead.

Pooling Support

I think a factory shouldn't provide pooling; it should be implemented on top of factory. Otherwise the simple factory interface also needs to deal with synchronization and return to pool semantics.

Support for Object Filters

When a factory instantiates an object, it can also pass the object to an object filter and return the output of the filter instead. This allows for the chaining of objects, which is similar to the servlet chaining popular in Web-based systems. It's fairly trivial to do, but adds a great value to the overall functionality. For example, using such a functionality you can convert an incoming result set to a comma-separated string or sum the values into a single entity. All this can be done without recompiling the target code. This properties file demonstrates how this can be accomplished:

```
Objects.employee.className=com.comp.Employee
Objects.employee.filterName=SumSalary
Objects.SumSalary.className=com.comp.Sum-
 EmployeeSal
```

The Default Object

When you're specifying target implementation objects via a properties file, it's possible that anything might go wrong in the construction process, resulting in an exception. In this case you'd like a default implementation to be available. For example:

```
MyInterface myObj = null;
try
{
 myObj = ObjectFactory.getObject
        (MyInterface.NAME,null)
}
catch( FactoryException x)
{
   x.printStackTrace():
   myObj = new MyDefaultObject;
}
```

Doing this everywhere discourages the programmer from using the facility. The following code alleviates the problem:

```
MyInterface myObj = ObjectFactory.get
 Object (MyInterface.NAME, args, new
 MyDefaultObject();
```

This method doesn't throw any exceptions, instead it returns the MyDefaultObject on any kind of internal exception, ensuring the default functionality of the application.

A Well-Defined Factory Interface to Fulfill the Factory Service

With that discussion we're able to define an interface for our factory object:

```
interface IFactory
{
   Object getObject(final String
interfaceName, Object args) throws
FactoryException;
   Object getObject(final String
interfaceName, Object args, Object
defaultObject);
}
```

An interface definition tells you only the contract of the factory and doesn't specify its intended behavior. For that reason I'm summarizing here the desired charac-

Using a simple application-level factory service, it's possible to effectively employ interfaces and implementations to write code that provides backward compatibility

teristics of a typical factory implementation:

1. Provides a static façade for easier calling semantics
2. Uses getObject, not createObject, semantics
3. Requires a default constructor for the objects to be constructed
4. Arguments are passed to the init method for initialization
5. Caching is based on the class name of the object
6. Caching is also based on the IsingleThreaded tagging interface
7. Supports object filters
8. Supports default objects
9. Uses an application-wide configuration interface for class definitions

ObjectFactory Implementation

Although we've defined an interface for a factory, we need a convenient set of static methods that would make use of the above interface internally. Such a facility provides the best of both worlds. On one side it obviates the need to instantiate a factory interface and on the other it adheres to a nonstatic factory interface definition. Here's one such implementation. Obviously many variations are possible, but the idea is to give the programmer the simplest way to accomplish the characteristics stated above (see Listing 1).

Role of an Application-Wide Configuration Service

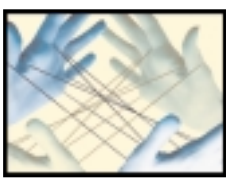
So far all the object definitions are stored in a properties file. It's not that difficult for the factory implementation to instantiate a properties object and read in the properties file. What if in the future you'd like to move the entries from a properties file to a database, an LDAP, or an XML configuration file?

For this reason the factory implementation accesses the object definitions through a common configuration interface.

```
Interface Iconfig
{
        public String getValue(final String key)
   throws ConfigException;
        public String getValue(final String key,
   String defaultValue );
}
```

Again, for programmer simplicity, this interface is exposed through a static Config class as follows:

```
Class Config
{
        static public String getValue(final String
   key) throws ConfigException;
        static public String
getValue(final String
   key, String defaultValue );
}
```



With this you'd be able to do:

```
String value = Config.getValue("key","default value");
```

Why go to such lengths when properties is such a simple interface; why not use it? Frequently you would want the IConfig implementation to exhibit the following characteristics:

1. Case-insensitive keys so you don't have to constantly remember capitalizations
2. Ability to include multiple-properties file inside the main configuration file
3. Ability to provide substitutions based on keys that were defined at the beginning of the properties file

All these can be provided gradually by the evolving implementations of IConfig. In this case, one of the facilities of the framework evolves using the same principles.

Evolving an Interface

So far I've discussed where the implementations are changing. What if the interface itself has migrated to the next level? Although less common, this will happen eventually, even in the best of designed interfaces. For instance, with the logging interface defined above, I'd like to extend the interface by providing a logging mechanism for exceptions. One approach is to extend the base interface instead of modifying it:

```
interface ILogFacility_1 extends
 ILogFacility
{
    public void log(Throwable t);
}
```

If there's an implementation that supports the ILogFacility_1 interface, it automatically supports the ILogInterface and hence can be used in a backward-compatible manner. On the other hand, if an older implementation that supports only ILogInterface is used in a newer environment that's expecting ILogFacility_1, there are two alternatives:

1. Let the system throw a class-cast exception and correct the mistake
2. Write the client code in such a way that this can be handled gracefully:

```
Object  logFacility =
 ObjectFactory.getObject
          (ILogFacility.NAME,null,new MyLogFacility);
if (logFacility instanceof ILogFacility_1)
          {
           (ILogFacility_1)logFacility.log(x);
          }
          else (logFacility instanceof ILogFacility)
          {
```

```
 (ILogFacility)logFacility.log-
(x.getMessage());
}
```

Request-Based Factories

As I've mentioned, factory plays a central role in adapting this evolution-based approach to programming. The factory I've discussed so far is primarily creation-based because it instantiates the necessary class and calls the well-known method to initialize it. A more sophisticated approach would be a request-based factory in which the input symbolic name refers to a request rather than a class name. You can simulate the creation-based factory from a request-based factory. For example:

```
Object o = RequestBasedFactory.getObject
   ("GET_OBJECT", null):
```

The request-based factory locates a class name identified by "GET_OBJECT" in a properties file. Instead of calling the init method to initialize the object, the request factory calls the "executeRequest" method with the parameters. Whatever the executeRequest method returns is then returned to the caller as the return object.

When the executeRequest method returns the self-reference, we basically have the RequestFactory behaving like a creation factory. RequestFactory has broader applications than a simple CreationFactory. For example:

```
RequestBasedFactory.getObject("GET_EMPLOYEE_
   ROWS","employee_name=John Doe")
```

invokes a database request executor identified by "GET_EMPLOYEE_ROWS" and returns the result set back to the caller.

Interfaces and Exceptions

I'd like to cover one more detail since we're dealing extensively with interfaces, particularly such generic interfaces as the factory interface I've shown here. When a factory tries to load a class at runtime and fails, all the client sees is the factory exception. If you're not careful, the root cause of the exception is never known. To deal with this scenario, interfaces should define exceptions that can carry with them a child exception, and the child exception can carry another child exception. This way the root exceptions are propagated all the way to the top. When the final exception is printed, you'll see a complete hierarchy of exceptions, which is extremely good for debugging purposes. I call this the *principle of Kangaroo exceptions*, and it's necessary for interface-based programming. This is because interfaces are like firewalls and compile-time exceptions can't propagate through them without help. This concept of Kangaroo exceptions hides the root exception in terms of the interface-specific exception.

Conclusion

Using a simple application-level factory service, it's possible to effectively employ interfaces and implementations to write code that provides backward compatibility. Even when you don't need this compatibility, it enables you to try new bug fixes without affecting the old behavior. This provides developers with some assurance that they won't undo everything they've done before.

Author Bio

Satya Komatineni is president of Active Intellect, Inc., and the author of a Java-based RAD framework for developing J2EE-based HTML applications (www.activeintellect.com/aspire/). He holds an MS in electrical engineering from the Indian Institute of Technology, New Delhi.

satya@activeintellect.com

Listing 1

```
class ObjectFactory implements IFactory
{
 private static m_self = new ObjectFactory();

 public static Object getObject(final String interfaceName, Object args) throws
FactoryException
 {
  return m_self.getObject(interfaceName, args);
 }
 public static Object getObject(final String interfaceName, Object args, Object
defaultObject)
 {
  return m_self.getObject(interfaceName, args);
 }

 // private non static methods
 Object getObject(final String interfaceName, Object args) throws FactoryException
 {
  // implement as appropriate
 }
 Object getObject(final String interfaceName, Object args, Object defaultObject)
 {
  // Implement as appropriate
 }

}
```

Download the Code! www.JavaDevelopersJournal.com

Optimization and JAVA

written by Irvin J. Lustig

No longer mutually exclusive

As Java continues to be used in a wider range of applications, one measure of its progress is in its ability to take on more robust, high-performance computational applications. One of the most demanding areas within commercial software is optimization. Best known as the driving technology for planning and scheduling inside supply chain management applications, optimization involves computational engines fueled by specialized algorithms that crunch through complex business problems to find quick answers that can save businesses enormous amounts of time and money.

Optimization applications have lived in a universe far distant from Java. They've largely been C-, C++-, and FORTRAN-based – just what you would expect from such sophisticated, computationally demanding software. However, rapid evolutionary forces in both Java and the optimization sector, combined with the relentless need for higher levels of intelligence in commercial software applications, are converging to the point where Java and optimization are no longer mutually exclusive.

Until quite recently Java programmers who wanted to include optimization within their applications needed to write their own Java wrappers around optimization code. Another limiting factor has been the lack of hands-on knowledge of high-end optimization techniques among developers, who often aren't skilled in model development and selecting the best-solving strategy. However, commercial Java Native Interface (JNI) wrappers for optimization code are now emerging, with pure Java-based development solutions imminent. In addition, recent ease-of-use breakthroughs in the form of tools for modeling and embedding optimization software are bringing this technology closer to the mainstream.

For Java developers, having access to optimization technology offers the potential for smarter applications that deliver greater efficiencies, not just in planning and scheduling – the bedrock of optimization – but in emerging areas such as customer-facing software for solutions such as Web self-service. This article introduces optimization to the Java programming community: what it is, what it isn't, and how it can be an invaluable tool for creating a wide range of applications, particularly for the Web. Java developers should come away from this article with a better understanding of optimization and how they might leverage this powerful and flexible technology in their own applications.

Definition of Optimization

Modern-day optimization is defined by the application of mathematical and constraint programming. However, the ideas of optimization go back to Isaac Newton and calculus. Every calculus student learns how to find the minimum value of a function of variables by taking the derivative and solving a set of equations for the answer.

Today, optimization methods represent a set of techniques for finding the best usage of limited resources. The best usage could be minimum cost, maximum profit, highest level of service, or quickest delivery time. The types of resources that may be limited are people, vehicles, equipment, time, supplies, and money. Optimization is applied to manufacturing systems, telecommunication networks, asset allocation, and other business processes where it's possible to come up with a mathematical model for the underlying business processes. The successful application of optimization has the following features:

- Represents decisions by using a set of decision variables in which the possible values of the decision variables are known
- States a set of constraints that the decision variables must satisfy
- States an objective function of the decision variables

Given this broad description, a variety of powerful algorithms have been developed that are able to determine solutions to such problems. With the advent of improved computational hardware power combined with the improvements in the underlying algorithms for solving optimization problems, it's now possible to solve problems in seconds on desktop computers that only 10 years ago required weeks of computing time. Because of this, the range of possible applications of optimization has widened.

VM Gear

www.vmgear.com

Optimization problems are inherently difficult to solve because the number of possible solutions to any particular problem is enormous. The most classic optimization problem is the traveling salesman problem: given a set of cities and starting at a home city, determine the shortest path that would enable the salesman to visit all the cities without visiting any city more than once, then returning home at the end of the tour. For a problem with 1,000 cities, there are over 102,567 possible solutions to consider! Sophisticated optimization algorithms are able to efficiently solve similar practical problems, despite the large number of potential solutions.

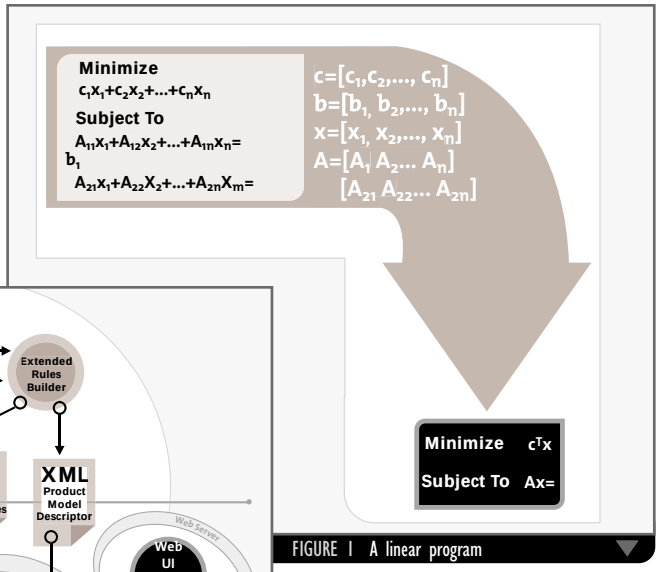


FIGURE 1 A linear program

Optimization Examples

Some examples help illustrate the kinds of problems that can be solved with optimization technology. For each of these problems the concepts of "best usage," stated as an objective, and "limited resource," reflected in constraints, will be described.

- An airline has a schedule of flights and a fleet of different aircraft types. The organization needs to determine the assignment of the aircraft to the different flights so as to maximize profit, while making sure that aircraft are scheduled for maintenance, have sufficient turnaround time between flights, and anticipated demand is met. Here, the "best usage" is the profit, as measured by the expected revenue obtained by assigning a particular aircraft type to a particular flight, less the cost of purchasing and operating enough aircraft to fly the entire schedule. The limited resources are the planes and the limits imposed by the airline's schedule. Optimization technology is used to determine the best assignment.
- An automobile manufacturer needs to sequence cars through a paint shop. Each car is painted a certain color. Each time the paint color is changed, an expensive paint purging operation must occur. The goal is to reduce costs and increase throughput by reducing the number of purges. Here the limited resources are the paint shop and the space allocated for cars waiting to be painted, and the objective is to find the best usage of the shop. Optimization is applied to find a sequence of cars that minimizes the number of times the paint system must be purged by grouping cars of the same color together and respecting the availability of the limited size of the waiting area.
- A consumer desires to obtain a home equity loan. Each consumer may have different objectives, such as minimizing payments, reducing overall debt, or maximizing the cash available from his or her home. At the same time, a bank has limits on the type of loan it's willing to approve. Optimization allows the bank to offer the loan best tailored for the consumer's needs by searching among all the possible loan configurations.

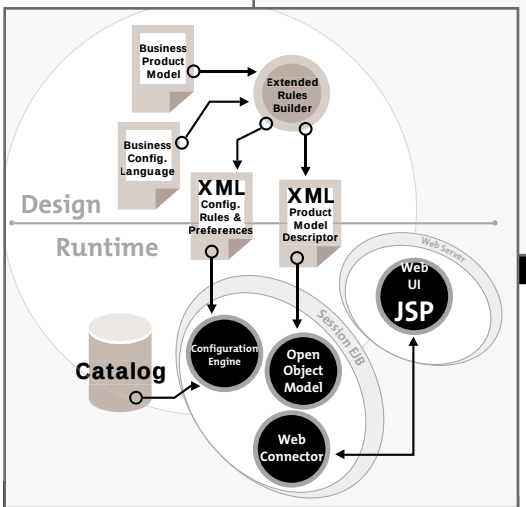


FIGURE 2 Java-based configuration tool

Optimization Technologies

While the foundations of mathematical optimization are found in calculus, the age of modern optimization began with the discovery of the simplex method for linear programming by George Dantzig in 1947, which led to the development of the fields of operations research and mathematical programming. Subsequently, developments in the artificial intelligence community led to the emergence of constraint programming as a technique that could be used to solve optimization problems. Today, numerous companies solve optimization problems using both techniques.

Mathematical Programming

Many optimization problems can be stated using decision variables that are real-valued, combined with objective functions that are linear and constraints that are linear inequalities. Such problems are known as linear programs. Here the word *program* refers to the original statement of the problem by Dantzig as a *programming problem*, where the problems Dantzig studied involved finding the best activities program for achieving a desired goal. If you restrict some or all of the decision variables to be integer valued, the problem is then an integer program.

Linear and integer programming are the foundations of the field of mathematical programming and are taught in many universities in operations research and industrial engineering departments, as well as in quantitative courses in many MBA programs. A linear program can be represented using numerical matrices and vectors. Figure 1 illustrates the algebraic and matrix-vector forms of a linear program.

In general, mathematical programming technologies are used for long-range planning applications. They're one of the keys behind the success and growth of supply chain management applications. Mathematical programming techniques are also used by all the major airlines for fleet assignment and crew scheduling. Telecommunication companies use mathematical programming techniques to optimally design their networks. The applications' growth has been driven by the improvement in algorithms and hardware so that linear and integer programs are solved by computationally intensive algorithms in a reasonable amount of time.

The underlying algorithms used to solve linear and integer programs require a substantial amount of floating point computation. Efficient implementations take advantage of the underlying processor and cache architectures, as well as compilers that can produce code that's efficient for a particular class of hardware. Near term, pure Java implementations of these algorithms won't deliver the required performance for all kinds of optimization applications. As an initial remedy, it's now possible to obtain JNI wrappers around the implementations of these algorithms, which allows Java developers to develop in Java while achieving the performance delivered by the algorithms.

Constraint Programming

Constraint programming has its roots in the research of the artificial intelligence community. Here, the word *programming* refers to a particular methodology for doing computer programming. To apply constraint programming a specialized programming language is required, or special libraries built into a mainstream programming language can be used. Abstractly speaking, a programmer lists variables of a computer program along with their domains of allowable values. Then, constraints are listed that the variables must satisfy. As opposed to linear programming, they can be in any form that the underlying constraint programming language allows.

The final step in applying constraint programming is to write a search strategy in the programming language indicating how the variables should be modified in order to find values of the variables that satisfy the constraints and, optionally, optimize an objective function. Sophisticated constraint propagation and domain reduction algorithms are used to process each constraint type so as to make the overall search strategy efficient. Listing 1 illustrates an example of a constraint program for coloring a map in the optimization programming language (OPL). The decision variables are represented by the array color, where for each country a specific color must be chosen. The constraints indicate which countries can't have the same color.

Recently, constraint programming has been successfully applied to optimization problems that appear in more operational and tactical applications, where real-time answers to problems are required and mathematical programming techniques are insufficient. Typically, these problems involve logical constraints on integer variables, which can be processed efficiently by a constraint programming engine.

Constraint programming has been successfully applied to scheduling, sequencing, and configuration problems. In supply chain management applications, it's used to determine the schedule of how products should be manufactured on a limited number of machines. The auto paint shop application mentioned earlier is solved using constraint programming techniques.

Finally, the loan configuration problem represents another successful application of constraint programming due to its ability to efficiently process the complex underwriting constraints that describe the loans in order to search for the best loan configuration for a customer.

Modeling Languages

A model of the underlying processes must first be built to successfully apply optimization techniques to solving business problems. In mathematical programming this has typically been done by writing software to generate the matrices representing linear and integer programming. More recently, specialized mathematical programming modeling languages have been developed that provide the capability to succinctly represent a mathematical programming problem. The initial development of these languages created stand-alone tools that would allow users to enter problems in the modeling language, integrate external data, and then solve the resulting program. Today, these languages are now provided in the form of libraries that allow easy integration of mathematical programming models into large-scale applications.

Constraint programming development has occurred in two directions. One was the development of special purpose languages for applying constraint programming techniques. Like mathematical programming languages, the specialized constraint programming languages were created as stand-alone tools and are now available as object libraries. The second direction is the creation of specialized constraint programming libraries on top of existing programming languages such as C++ and Java.

Java-constraint programming tools are still in a state of development. The next year should see the introduction of pure Java constraint programming engines.

The optimization programming language is unique in that it provides a language that enables the use of both mathematical and constraint programming techniques. An example of a linear program written in OPL is shown in Listing 2. The data is represented by the sets Products and Resources, and the arrays product and capacity. The decision variables are the arrays inside and outside. The modeling language representation directly maps to the matrix/vector notation discussed earlier.

Java Tools for Optimization

Optimization has been remote to the Java community mainly due to the lack of effective tools to utilize it. Affecting the technology's wider spread into the Java community has been the inherently slower performance of the Java platforms that made fully leveraging the technology difficult. But things are changing. New technology advances and new features in optimization software mean easier access for Java programmers and better performance. Complementing these advances and making it relevant are the performance improvements to the Java platforms that help minimize the performance hit for optimization applications.

As mentioned above, traditional optimization as represented by mathematical programming applications have yet to be offered directly on the Java platform due to performance considerations. Today, Java interfaces to existing engines are now available, providing the benefits of a Java interface to a high-performance computational engine. If demand emerges for pure Java versions of mathematical programming tools, it's likely such solutions will be developed. Fortunately, the speed of the underlying engines has improved to a point where "Java" optimization can be used in real-time applications. In particular, marketplace auctions on the Web represent an excellent application of this technology.

An interesting development for both mathematical and constraint programming is the introduction of ILOG OPL Studio 3.5, which allows models represented in OPL to be embedded inside Java programs. A Java developer assembles the required inputs for an OPL model and passes the data and the model description to the OPL component libraries, which are supplied with JNI wrappers. The libraries then interpret the model and the data and solve the resulting optimization problem. The optimal values of the decision variables can then be obtained by querying methods in the OPL component libraries.

The advantage of using JNI in this case is that the underlying performance of the optimization engine on different hardware platforms is preserved. Furthermore, the process of developing an optimization model, which requires expertise in understanding the underlying business processes, is separated from the process of integrating the optimization model into a Java program.

On the other hand, Java versions of constraint programming tools can be useful in a new class of Web-driven applications that are based on solving configuration problems, such as the loan configuration problem described earlier. In fact, the first versions of these tools will be available later this year. To illustrate, let's consider the problem of configuring an automobile that's subject to the requirements of a consumer and the constraints determined by the automobile manufacturer. The consumer may want the lowest-priced car with specific features. The manufacturer may have constraints that indicate the compatibility of certain options. For example, certain models may not be available with a sunroof or a particular engine. Constraint programming can be used to represent the possibility of different choices and the constraints on these choices.

On the Web a form can be used to represent the different choices that are available to the consumer for configuring his or her automobile. After the consumer makes a choice, the constraint programming engine can then eliminate incompatible choices. Once a set of choices is made by the consumer, the choices left open by the user can then be considered for optimization by the constraint programming engine, and the lowest-priced car subject to the consumer's choices and the manufacturer's constraints can be determined.

For this application a Java-based constraint engine is required, and it can be easily integrated within the J2EE framework. Figure 2 illustrates the design of such an application. To describe a configuration problem at design time, a business product model and configuration language are used with a builder, which generates an XML representation. This is then used at runtime.

Constraint programming is used inside the configuration engine. Because the state of the current configuration must be represented and updated quickly, using a non-Java engine will be problematic. The performance improvements in the Java 2 framework should allow this kind of application to execute efficiently within a Web-based environment.

Conclusion

As more real-time decisions are required by businesses, optimization applications will appear on the Web and integration with Java will be a requirement. Java programmers are already creating JNI interfaces to existing optimization engines to allow access to Java programmers. Whether it's by using JNI interfaces to high-performance engines or by using pure Java applications, a growth of such applications should occur within the next couple of years. Fueling this growth will be the improvements in computer hardware and the underlying algorithms for optimization, allowing businesses to reduce costs and increase profits. The opportunities are unlimited.

Author Bio

Dr. Irvin Lustig is ILOG's optimization evangelist responsible for education and marketing initiatives around optimization. He received his PhD in operations research from Stanford University and a ScB in applied mathematics/computer science from Brown University. He's also the author of more than 30 scientific research papers.

ilustig@ilog.com

Listing 1

```
enum Country {Belgium,Denmark,France,
              Germany,Netherlands,Luxembourg};
enum Colors {blue,red,yellow,gray};
var Colors color[Country];
solve {
      color[France] <> color[Belgium];
      color[France] <> color[Luxembourg];
      color[France] <> color[Germany];
      color[Luxembourg] <> color[Germany];
      color[Luxembourg] <> color[Belgium];
      color[Belgium] <> color[Netherlands];
      color[Belgium] <> color[Germany];
      color[Germany] <> color[Netherlands];
      color[Germany] <> color[Denmark];
```

Listing 2

```
enum Products ...;
enum Resources ...;

struct ProductData {
   float+ demand;
   float+ insideCost;
   float+ outsideCost;
   float+ consumption[Resources];
};

ProductData product[Products] = ...;
float+ capacity[Resources] = ...;

var float+ inside[Products];
var float+ outside[Products];

minimize
   sum(p in Products) (product[p].insideCost*inside[p] +

product[p].outsideCost*outside[p])
subject to {
   forall(r in Resources)
      sum(p in Products) product[p].consumption[r] *
inside[p]
            <= capacity[r];

   forall(p in Products)
      inside[p] + outside[p] >= product[p].demand;
};
```

Download the Code! www.JavaDevelopersJournal.com

using javabeans in DREAMWEAVER ULTRADEV 4

Make your workflow process more productive and effective

written by MICHAEL BARBARELLI
and DAVID DEMING

Given the publication this article is appearing in, I'm not going to spend much time trying to sell you on the benefits of the Java platform for Web application development. Let's assume we're all on the same page here.

I think we can also agree that developing Web applications using JavaServer Pages (JSP) could be easier. The technology builds on the rich development architecture of the Java platform, but there aren't many programs that effectively mask the complexity of the language to simplify Web application development.

Macromedia Dreamweaver UltraDev 4 simplifies visual Web application authoring using JSP while taking advantage of the unique capabilities of the Java platform. This includes database connectivity through JDBC, connecting to other back-end and legacy systems through JavaBeans, and supporting custom tag libraries. In this article we explore how UltraDev can help Web application development teams (or individuals) streamline the process of deploying applications. We'll use JavaBeans as an example technology to illustrate this point.

A little more marketing background and then I promise I'll get technical and dig into how UltraDev makes it easier to incorporate JavaBeans into your Web development workflow. It's built on the core architecture and shares all the features available in Macromedia Dreamweaver, a visual Web authoring environment for professional Web developers. The product is fully customizable and scriptable using JavaScript, XML, and HTML. This means that with UltraDev you get an HTML editor to help visualize your design.

Now that you have a basic background on UltraDev, let's look at how JavaBeans can be used with it.

As you know, the Java 2 Platform, Enterprise Edition includes JSP. The Java platform allows access to a varied range of APIs. The JavaBean API coupled with JSP helps users to develop Web applications by moving much of the server-side logic into the bean and focusing the JSP page on the presentation layer. By developing JavaBeans to contain most of your server-side logic, reusable classes can be defined, greatly reducing future application maintenance requirements. This makes it possible to extend those classes and tailor them to different requirements.

What this means to most Web development teams today is that a wide variety of skills are necessary to effectively deploy Java-based Web applications. You need Java programming experts to develop the JavaBeans, design and layout experts to create the front-end user interface (usually in HTML), and Web application programmers to glue the back-end logic and data structure to the front-end user interface. Although, depending on the size of the team, some of these tasks might be performed by the same people, getting all of them running together smoothly is a challenge.

I think most people involved in Web application development can relate to the following scenario. A front-end designer creates a static user interface; at this point the back-end system (database data business logic, etc.,) may not be designed, or it may not be accessible enough to integrate with the user interface. Once it is ready, the Web application developer must then go through an iterative process of plugging in the back-end system to the user interface, uploading the result to a test server, determining what works and what doesn't, and asking the designer to tweak the design to fit the data structure. Where UltraDev really helps is in this test/tweak/redesign process.

One of the most difficult aspects of this problem is that it's hard to find a single person, let alone a single application, that can effectively tweak design elements and plug in server-side logic and dynamic data. As an application development environment, UltraDev can help tremendously here.

Let's assume we're responsible for the task I just described. An enterprise Java developer has given us a JavaBean that masks the complexity of a set of database functions that store customer information. We must now integrate the data from that bean into an HTML user interface.

Using JavaBeans in a Code-Only Environment

When doing tasks like this by hand, the Web application developer needs to know a bit of JSP syntax to effectively incorporate a JavaBean into the design. Assuming we have a bean called *Customers* available, our Web application developer would include a line in the HTML that looked something like this:

```
<JSP:useBean id="Customers" class="Data.Customers" />
```

This creates an instance of the bean that's then available on the rest of the Web page. The next step might be to set certain properties of the bean to ensure we're accessing the proper customer. This is one of the beauties of JavaBeans – the application developer doesn't need to know anything about the structure of the database that stores the information about the customer, just the unique ID that's used to identify them.

```
<JSP:setProperty name="Customers" property="ID"
value="<%= request.getParameter("ID") %>"/>
```

In this case things became even more complicated – the ID for the customer was stored in a form variable called ID that came across in the request.

Once this property is set, the application developer can then plug in various customer properties at appropriate points in the design. Let's assume the designer had earmarked a portion of the page to contain a personalized greeting. The application developer would wade through the HTML to locate the appropriate area. A message welcoming the customer by name could then be inserted, for example:

```
<B>Welcome to our site, <JSP:getProperty
name="Customers" property="firstName"/></B>
```

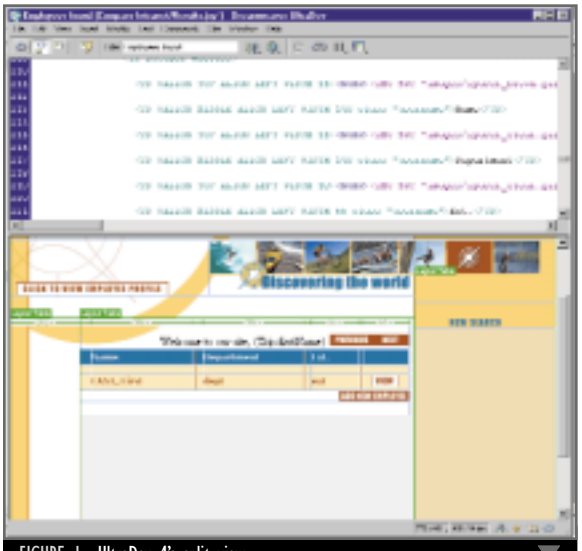



FIGURE 1 UltraDev 4's split view

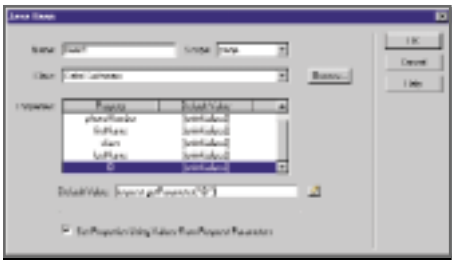


FIGURE 2 The UltraDev Bean initialization dialog

The application developer just inserted HTML and JSP code to create a personalized greeting for each customer.

The last, often overlooked, step in this workflow is for the application developer to upload this completed file to the application server and preview it in the browser. It's usually at this point where things get difficult. Although it might work perfectly in this simple case, rarely in the real world will the initial design perfectly accommodate all the dynamic elements needed on the page. This is where the dance of transferring files back and forth between the developer and the layout designer leads to delays and frustration in getting the design and dynamic elements just right.

Designers get frustrated because they can't see the dynamic data on the page as they're tweaking the layout – they have to work with placeholders. They have to take the developers' word for it – "That one table cell isn't quite big enough. You'll have to move that image somewhere else…." And application developers get frustrated because designers often step all over their hand-developed code when tweaking the design. This back and forth can be a serious headache, even when both tasks are performed by the same person! UltraDev can offer a compelling solution to this problem.

Using JavaBeans in UltraDev's Visual Development Environment

Because hand coding is so important to most developers, it would be possible to implement the above scenario exactly as described using UltraDev. By operating only in code-view, the application developer could diligently type out each of the above lines of code in the appropriate places in the HTML layout. Of course, part of the power of UltraDev is making that process easier and more effective.

One of the first things a traditional coder will notice about UltraDev is all the visual panels and inspectors that surround the target document to be edited. Assuming that the flat HTML file is open and ready to be modified with dynamic data, the user also has a choice in how to view it. There are three editing modes in UltraDev that can be used depending on the situation

and preference of the developer. Code view shows all the code behind the page (HTML and Java) using syntax-coloring for clarification. Design view shows a visual representation of what the page will look like when viewed in the browser, while split view allows both the code and visual representation of the page to be viewed and edited at once (see Figure 1).

Let's assume we've chosen to work in design view. The first step with UltraDev is to set up the JavaBean as a data source. This process consists of browsing to the appropriate class file, typing in the initialization parameters, and giving the bean a name. You can even visually set any of the available properties of the bean at this point (see Figure 2). This is one benefit of the visual environment – it instantly lets you know what properties are available to set. This is where we could type in the ID property for the customer, thus eliminating the need for a setProperty tag.

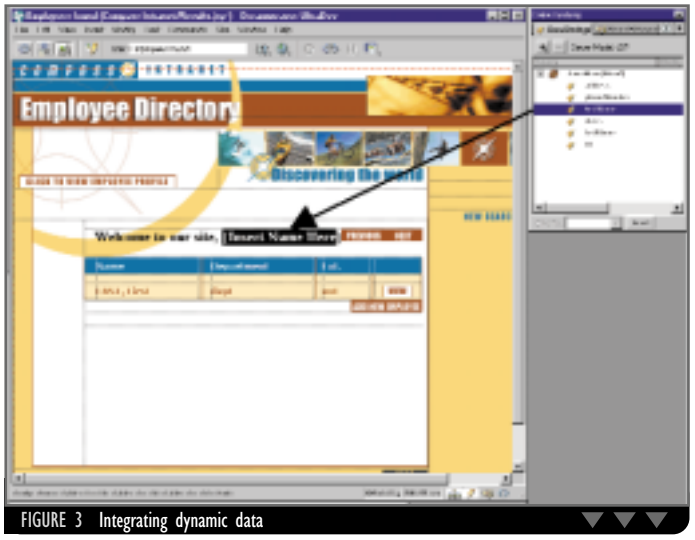


FIGURE 3 Integrating dynamic data

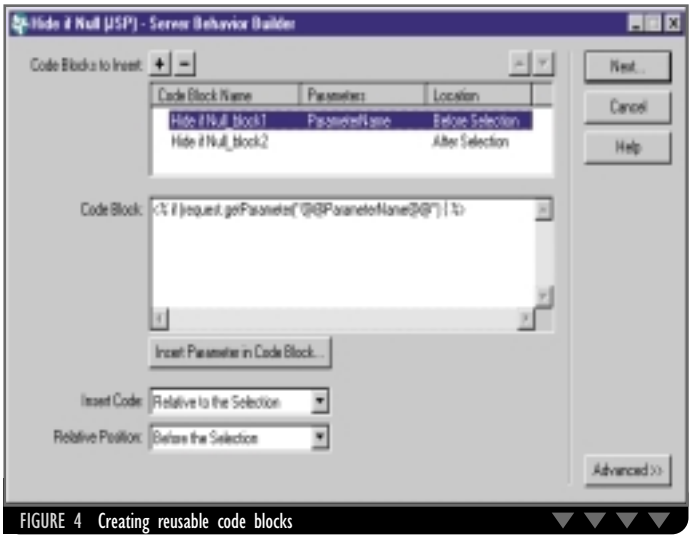


FIGURE 4 Creating reusable code blocks

FIGURE 5 Applying a behavior from the server behavior menu

Once all this is done, UltraDev automatically uses introspection to discover all the available setter and getter methods of the bean and shows them in the data bindings inspector. To use these elements on the page, our application developer has to simply drag and drop the various elements into place. In Figure 3 the designer notices the placeholder on the page and can easily replace it with the dynamic bean element. UltraDev generates the necessary JSP code behind the scenes, creating the dynamic welcome message that includes the individual customer name from the bean.

Let's go beyond our simple example for a moment and assume the app builder needs to build more complicated behavior into the page. For example, the developer wants to hide the entire greeting message if the user's first name is not available, that is, it was never passed from the form on the previous page. Since this is such a common task, entering the code by hand each time is laborious. Using the UltraDev server behavior builder, the developer could automate this process by teaching UltraDev the code that should be used in this situation and setting the appropriate variables to make it work. Then this behavior would be saved and used later, automating this task. Let's work through this example:

- The developer opens UltraDev's server behavior builder and gives this new behavior a name, "Hide if null".
- The developer then defines the first code block to come before the area of the page that is to be hidden:

```
<% if (request.getParameter("@@Parameter@@") == null)
{ %>
```

The "@@" surrounding Parameter tells UltraDev that this is a variable name that should be entered by the user each time the server behavior is used.

- The developer configures the first code block to come before the area of the page the user had selected.
- The developer defines the second code block to come after the user's selection:

```
<% } %>
```

- The developer determines how the user enters the parameter when using the behavior. In this case we'll have the user enter it using plain text (see Figure 4).

Now that the developer has saved this behavior, it can be easily accessed again though the server-behavior menu. To have a certain area of the page hidden if a request variable is null, the developer needs to select the area on the page, click on the server behavior, and type in the appropriate request parameter (see Figure 5). The developer can even share this with other users in the group to help them speed their development process.

Instead of saving the file, uploading it, and then previewing it on the browser, the application developer can use UltraDev's LiveData Mode to see instantly what this design is going to look like in the browser (see Figure 6). With this instant visual feedback the application builder can then see the effects that different text formatting will have on the design, and even make layout tweaks visually to accommodate the dynamic data (with the designer's approval, of course!).

If the file needed to be passed back to the designer for more complicated changes, that designer could work in UltraDev's LiveData Mode to visually edit the HTML layout and instantly see how the design changes affect the page that incorporated the dynamic data. They can also instantly tell if their design change has broken some aspect of the code because UltraDev will give

FIGURE 6 Using UltraDev's LiveData Mode

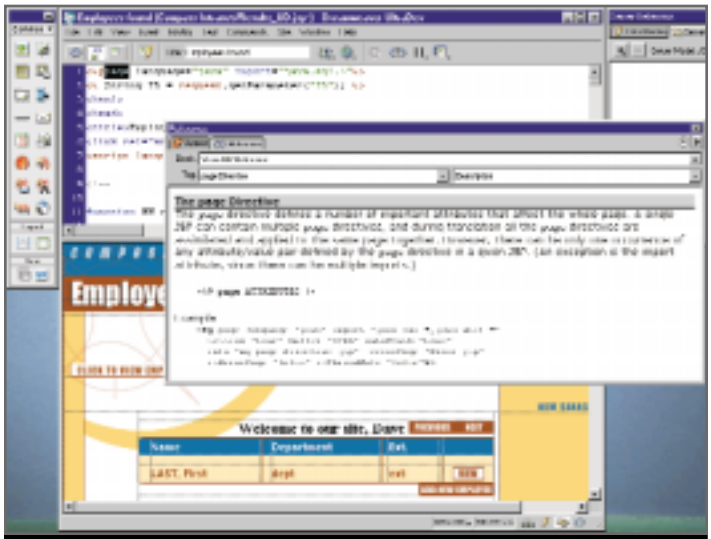


FIGURE 7 The embedded Wrox JSP reference

them an error message in LiveData Mode. They can then use the unlimited undo feature to return the document to its previous state before things went awry. This saves developers and designers lots of heartache by eliminating the possibility that the designer could inadvertently break the code of the page during a design change.

More enterprising designers can even use the code/design split view to see what code is generated by each of the dynamic elements on the page. This has been used as an effective learning tool for designers to become familiar with JSP style code and what it does. Anything that's unclear can be instantly looked up in the Wrox JSP reference book that's built right into the product (see Figure 7).

The idea here is that the instant visual feedback UltraDev provides makes this whole workflow process more productive and effective. It cuts down on the amount of time that designers and developers have to spend tweaking their work to accommodate what the other is doing. While UltraDev is certainly not a replacement for a Java development environment that's used to create JavaBeans and other components without visual elements, it's absolutely crucial for streamlining the process of combining a visual interface with back-end systems. Using UltraDev, teams of application development professionals can streamline their processes, allowing them to spend more time optimizing their design and code.

Author Bios

Michael Barbarelli, lead technician for Macromedia's UltraDev technical support group, is responsible for managing the product from the customer's viewpoint. He studied computer science at the University of California.

David Deming is the product manager for Dreamweaver UltraDev. He has a mcechanical engineering degree from UC–Berkeley.

mbarbarelli@macromedia.com

ddeming@macromedia.com

Breeze Factor

www.breezefactor.com

A Practical Solution for the Deployment of JavaServer Pages

Download Web apps without compromising your security

WRITTEN BY
JOHN GOODSON

While Java databases have become essential for Web applications, developing performance-oriented Java Database Connectivity (JDBC) applications can be challenging. But by following tried-and-true approaches, it is possible to develop and fine-tune JDBC applications to make them run faster, jump higher, and make fewer trips to the database.

Below, we present practical guidelines for many of the common JDBC applications.

GUIDELINE 1: *USE METADATA WITH CARE*

The first guideline is to be careful about using metadata. Although it's almost impossible to write JDBC applications without database metadata methods, you can improve system performance by minimizing their use.

Cache Information to Limit Executions

To return all result-column information mandated by the JDBC specification, a JDBC driver may have to perform multiple queries, joins, and subqueries in order to return the necessary result set for a single call to a database metadata method. Applications should cache information returned from result sets that generate database metadata methods so that multiple executions are unnecessary. For example, call getTypeInfo once in the application and cache the elements of the result set that your application depends on. It is unlikely that an application will use all elements of the result set generated by a database metadata method, so the cache of information should not be difficult to maintain.

Avoid Null Search Patterns

One thing to avoid with metadata is the use of null arguments or search patterns with database metadata methods because this generates time-consuming queries. In addition, network traffic can increase due to unnecessary result set information. To avoid these problems, always supply as many non-null arguments as possible to result sets that generate database metadata methods.

Moreover, because these methods are slow, applications should invoke them efficiently. Many applications pass the minimum number of non-null arguments required for the function to return a successful result set.

For example:

```
ResultSet WSrs = WSc.getTables
(null, null, "WSTable", null);
```

should be:

```
ResultSet WSrs = WSc.getTables
("cat1", "johng", "WSTable",
"TABLE");
```

Unfortunately, sometimes you don't know much about the object for which you are requesting information. But any information that the application can send the driver when calling database metadata methods can result in improved performance and reliability.

Use Dummy Queries

Avoid using getColumns to determine characteristics about a table. Instead, use a dummy query with getMetadata.

Consider an application that allows the user to choose the columns. Should the application use getColumns to return information about the columns to the user or prepare a dummy query and call getMetadata? Look at Cases 1 and 2 on the next page for the answer.

GUIDELINE 2: *RETRIEVE DATA EFFICIENTLY*

Too many calls from the application to the driver can cause your application to run more slowly. To improve performance, try to reduce the size of data that is returned. Also, avoid retrieving long data unless it is absolutely necessary. Retrieving long data across the network is slow and resource-intensive. Moreover, most users don't want to see long data. If a user does want to see these result items, the application can query the database again, specifying only the long columns in the select list. This method allows the average user to retrieve the result set without having to pay a performance penalty of higher network traffic.

While it's best to exclude long data from the select list whenever possible, some applications do not formulate the select list before sending the query to the JDBC driver. (For example, some applications select * from <table name>....) If the select list contains long data, the driver must retrieve that data at fetch time, even if the application doesn't bind the long data in the result set. In this scenario, whenever possible use a method that does not retrieve all columns of the table.

To allow the application to control how long data is retrieved in the application, designers use the getClob and getBlob methods. However, in many cases the JDBC driver emulates these methods due to the lack of true locator support in the DBMS. In these cases the driver must retrieve all of the long data across the network before exposing the getClob and getBlob methods.

Sometimes retrieving long data is necessary. When this is the case, remember that most users don't want to see 100KB, or more, of text on the screen.

GUIDELINE 3: *REDUCE THE SIZE OF DATA RETRIEVED*

One option for reducing network traffic and improving performance is to reduce the size of any data being retrieved to some manageable limit. You can do this by calling setMaxRows, setMaxFieldSize, and the driver-specific SetFetchSize. Another way to reduce the size of the data being retrieved is to decrease the column size. If the driver allows you to define the packet size, use the smallest size that will meet your needs.

GUIDELINE 4: *USE MARKERS TO CALL STORED PROCEDURES*

One of the most complex tasks in JDBC programming is the handling of stored procedure calls.

JDBC drivers can call stored procedures on the database server either by executing the procedure as any other SQL query or by optimizing the execution by invoking a Remote Procedure Call (RPC) directly into the database server. Instead of using literal arguments, always use parameter markers for the argument markers when calling stored procedures.

Executing the stored procedure as a SQL query results in the database server parsing the statement, validating the argument types, and converting the arguments into the correct data types. Remember that SQL is always sent to the database server as a character string. For example, look at this command: "{call getCustName (12345)}." In this case, even though you might assume that the argument to getCustName is an integer, the argument is actually passed inside a character string to the server. The database server would parse the SQL query, isolate the single argument value 12345, and then convert the string "12345" into an integer value.

By invoking an RPC inside the database server, you avoid the overhead of using a SQL character string. Instead, the procedure is called by name with the argument values already encoded into their native data types. Here are two examples:

Example 1

Stored procedure cannot be optimized to use a server-side RPC. The database server must parse the statement, validate the argument types, and convert the arguments into the correct data types.

```
CallableStatement cstmt =
conn.prepareCall ("call getCustName
(12345)");
ResultSet rs = cstmt.executeQuery
();
```

Example 2

Stored procedure can be optimized to use a server-side RPC. The application calls the procedure by name and the argument values are already encoded, so the load on the database server is less.

```
CallableStatement cstmt –
conn.prepareCall ("Call getCustName
(?)");
cstmt.setLong (1,12345);
ResultSet rs=cstmt.executeQuery();
```

GUIDELINE 5: *OPTIMIZE JDBC PERFORMANCE THROUGH SMART DISK I/O*

You can improve application performance by limiting the amount of input/output. Connections and transactions are a good place to start looking for problems.

Managing Connections

Connection management has a direct effect on application performance. So design your application to connect once and use multiple statement objects, instead of performing multiple connections. Also, avoid connecting to a data source after establishing an initial connection.

Although it's a good thing to gather driver information at connect time, don't minimize the effect by connecting twice. For example, some applications establish a connection and then call a method in a separate component that reattaches and gathers information about the driver. Applications that are designed as separate entities should pass the established connection object to the data collection routine instead of establishing a second connection.

Besides connecting twice, another bad practice is to connect and disconnect repeatedly throughout your application to perform SQL statements. Connection objects can have multiple statement objects associated with them. Statement objects, which are defined as memory storage for information about SQL statements, can manage multiple SQL statements.

CASE 1: *GETCOLUMNS METHOD*

```
ResultSet WSrc = WSc.getColumns
(... "UnknownTable" ...);
// This call to getColumns will
generate a query to
// the system catalogs... possi-
bly a join
// which must be prepared, exe-
cuted, and produce
// a result set
. . .
WSrc.next();
string Cname = getString(4);
. . .
// user must retrieve N rows from
the server
// N = # result columns of
UnknownTable
// result column information has
now been obtained
```

CASE 2: *GETMETADATA METHOD*

```
// prepare dummy query
PreparedStatement WSps =
WSc.prepareStatement
    (... "SELECT * from
UnknownTable WHERE 1 = 0" ...);
// query is never executed on the
server -
// only prepared
ResultSetMetaData
WSsmd=wsps.getMetaData();
int numcols =
WSrsmd.getColumnCount();
...
int ctype =
WSrsmd.getColumnType(n)
...
// result column information has
now been obtained
// Note we also know the column
ordering within the
// table!  This information can-
not be
// assumed from the getColumns
example.
```

In both cases, a query is sent to the server. But, in Case 1 the query must be evaluated and return a result set to the client, so Case 2 delivers better performance.

Pooling is another recommended practice for managing connections. You can improve performance significantly this way, especially for applications that connect over a network or through the Web. Connection pooling lets you reuse connections; closing connections does not close the physical connection to the database. When an application requests a connection, an active connection is

reused, thus avoiding the network I/O needed to create a new connection.

Managing Commits in Transactions

Committing transactions is extremely slow and disk I/O-intensive. Therefore, always turn Autocommit off by setting WSConnection.setAutoCommit(false). There are two reasons for this.

First, the database server must flush back to disk every data page that contains updated or new data. This is not a sequential write, but a searched write to replace existing data in the table. By default, Autocommit is on when connecting to a data source, and Autocommit mode usually impairs performance because of the significant amount of disk I/O needed to commit every operation.

Second, some database servers do not provide an Autocommit mode. For this server type, the JDBC driver must explicitly issue a COMMIT statement and a BEGIN TRANSACTION for every operation sent to the server. So, in addition to the large amount of disk I/O required to support Autocommit mode, a performance penalty is paid of up to three network requests for every statement issued by an application.

Using transactions can help application performance, but it's not a panacea. Leaving transactions active can reduce throughput by holding locks on rows for long times, preventing other users from accessing the rows. Commit transactions in intervals that allow maximum concurrency.

GUIDELINE 6: *CHOOSE THE RIGHT TRANSACTION MODEL*

Many systems support distributed transactions, that is, transactions that span multiple connections. But distributed transactions are at least four times slower than normal transactions due to the logging and network I/O necessary to communicate among all the components involved in the distributed transaction. So, unless distributed transactions are required, avoid using them. Instead, use local transactions whenever possible.

And for the best system performance, design the application to run under a single Connection object.

GUIDELINE 7. *USE GETBESTROWINDENTIFIER() FOR COLUMN UPDATES*

Use getBestRowIndentifier() to determine the optimal set of columns to use in the Where clause for updating data. Pseudo-columns often provide the fastest access to the data, and these columns can be determined only by using getBestRowIndentifier().

Some applications cannot be designed to take advantage of positional updates and deletes. These applications typically update data by forming a Where clause consisting of some subset of the column values returned in the result set. Some applications might formulate the Where clause by using all searchable result columns in calling getPrimaryKeys(), or in calling getIndexInfo() to find columns that might be part of a unique index. These methods typically work, but might result in complex queries.

Consider the following example:

```
ResultSet WSrs =
WSs.executeQuery
                    ("SELECT
first_name, last_name,
ssn, address, city, state,
zip

FROM emp");
// fetchdata
...
WSs.executeQuery ("UPDATE EMP SET
ADDRESS = ?
WHERE first_name = ? and last_name
= ? and ssn = ?
     and address = ? and city = ?
and state = ?
     and zip = ?");
// fairly complex query
```

Applications should call getBestRowIndentifier() to retrieve the most optimal set of columns (possibly a pseudo-column) that identifies any given record. Many databases support special columns that are not explicitly defined by the user in the table definition but are "hidden" columns of every table (for example, ROWID and TID). These pseudo-columns generally provide the fastest access to the data because they are pointers to the exact location of the record. But because pseudo-columns are not part of the explicit table definition, they are not returned from getColumns. The only method of determining if pseudo-columns exist is to call getBestRowIndentifier().

Consider the previous example again:

```
...
ResultSet WSrowid =
getBestRowIndentifier()
   (.... "emp", ...);
...
WSs.executeQuery ("UPDATE EMP SET
ADDRESS = ?
WHERE first_name = ? and last_name
= ? and ssn = ?
     and address = ? and city = ?
and state = ?
     and zip = ?");
// fastest access to the data!
```

If your data source does not contain special pseudo-columns, then the result set of getBestRowIndentifier() consists of the columns of the most optimal unique index on the specified table (if a unique index exists). Therefore, your application does not need to call getIndexInfo to find the smallest unique index.

Conclusion

With thoughtful design and implementation, you can improve the performance of your JDBC applications. By following the suggested guidelines in this article, your applications can run more efficiently and generate less network traffic. Review your JDBC applications to find hidden treasure that will boost performance and reduce network traffic.

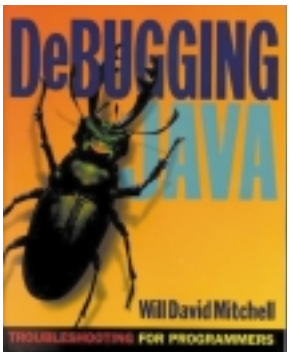
Author Bio

John Goodson is the vice president of research and development for MERANT DataDirect. For nearly ten years, John has worked closely with Sun and Microsoft on the development of database access standards, and is an active member of the Expert Panel for the JDBC specification evolution. John holds a BS in computer science from Virginia Tech.

john.goodson@merant.com

Debugging Java: Troubleshooting for Programmers

Learn Debugging Skills



BY WILL DAVID MITCHELL

PUBLISHED BY OBBORNE/MCGRAW HILL

REVIEWED BY JAMES MCGOVERN

In my travels I've run across lots of bad Java code (some of which I've written myself). Usually it's due to unrealistic project deadlines, bad estimates for how long something will take, no architecture in place, and developers' not really understanding the subtleties of the language.

Many people have learned Java by reading those 1,000-page books where you cut and paste code from the accompanying CD. Usually this teaches you how to get simple things working but doesn't help you learn Java. This book is the solution to all these problems.

The author, who has years of experience teaching computer science, emphasizes the importance of learning debugging skills. Industry research has shown that people who learn these skills first, master computer languages significantly faster. For the beginning developer, this helps you avoid picking up bad coding habits.

The book is 13 chapters, 460 pages, and reasonably priced at $39.99 ($31.99 on Amazon). Unlike many of the other Java books on the market, even the appendix provides useful information and is not filler. The text contains useful tips and advanced troubleshooting techniques for tracking down bugs. It also provides guidance on how to avoid scope creep and reduce deadline pressures by using proper estimation techniques.

Chapter 1 begins with a profound statement: "No amount of testing finds all bugs." There's no such thing as perfect code. With this in mind, the author guides you to use prioritization for finding and fixing bugs. Chapter 2 explains why bugs exist, which on the surface seems obvious (the other developers aren't always clueless). The author suggests that developers write documentation before writing code. The documentation will turn into a programming specification that can be reviewed by the business community, which in turn will help relieve pressure and buy you more time to develop properly.

Chapter 3 goes into actually practicing creating bugs so you'll be familiar with the outcomes such as exceptions or "not a number." Chapter 4 talks about risk factor analysis and provides great insight into figuring out how long something will take to develop. Chapter 5 discusses the usage of IDEs and brings up some very good recommendations. You'll learn how to use macros to reduce defective code. The author even points out how to use Microsoft Word as your IDE and integrate spell checking and error correction facilities.

I liked Chapters 6–8 the best. Chapter 6 covers the 80/20 rule, which says that 80% of your bugs are most likely to appear in 20% of your code. He notes additional tools for tracking down bugs that are syntactic in nature, and even suggests using LINT tools for finding them. For those who are not familiar with LINT tools, they're parsers that identify potential syntax errors in code that will compile but may lead to problems. Its origin comes from the C/C++ world. For instance, a LINT tool will flag the following code as a potential error: String abc = "\0123"; it will say you may have tried to incorrectly specify an Octal value. Other checks would include whether a loop should start with 0 or 1, using = in place of ==, memory leaks, thread conflicts, and Boolean checks.

Chapter 7 talks about best practices, which will further help you develop good coding habits. Chapter 8 discusses automated debugging and using debugging tools such as Sitraka Software's JProbe.

Chapter 9 covers what the author calls "Grand Debugging Strategies," which will give you insight on how to avoid regression errors. Chapter 10 covers black box testing. Black box testing checks a particular class's functionality by determining if its public interface performs according to specification without regard to implementation details.

Chapter 11 is obligatory reading for anyone developing threaded applications. You'll learn when and where to make your methods synchronized to avoid corrupted state. Synchronized methods carry a heavy performance penalty in Java and usually execute at 25% of the speed relative to a non-synchronized method. You'll learn how to make your applications faster.

The last two chapters are final thoughts about political correctness as well as conducting other forms of testing, such as stress and fault insertion. There's plenty of humor in both of these chapters as they even give you an idea on how to get your users to think they did something wrong when hitting a bug in an application.

Overall, this is a high-quality, easy-to-read book that promotes learning. I recommend it for both beginning as well as advanced Java developers. In fact, I'd suggest every project manager buy a copy of this book for his or her team members and make it mandatory reading. This book definitely rates five stars.

james.mcgovern@htsco.com

AUTHOR BIO

James McGovern is an enterprise architect for Hartford Technology Services Company, L.L.C., an information technology consulting and services firm dedicated to helping businesses gain competitive advantage through the use of technology. He's a coauthor of multiple books on Java and a member of several Internet advisory boards.

Dates and Calendars

Dealing with pure dates using the BigDate class

WRITTEN BY
ROEDY GREEN

Dealing with dates and times is probably the most confusing aspect of Java for newbies. There are three reasons for this:

1. The date and time classes are the most poorly designed of all the Sun class libraries.
2. The standard class libraries force you to deal with time zone and time of day, even when they're irrelevant to your problem.
3. The vocabulary used in the various date classes is inconsistent.

Vocabulary

- ***Day of month:*** The day of the month 1.31, sometimes called the date.
- ***Day of week:*** Day of the week for a given date; for example, Sunday=1 ...Saturday=7.
- ***DST offset (daylight saving time offset):*** The number of millisecond correction to account for daylight saving time; 0 if daylight saving time is not in effect for the time stamp specified. If a one-hour daylight saving is in effect, the offset will be 3,600,000. Add both the DST and the zone offsets to UTC to get local time.
- ***ISO day of week:*** Day of the week 1 to 7 for this date according to the ISO standard IS-8601. Monday=1...Sunday=7.
- ***ISO week number:*** Week number 1 to 53 of the year that this date falls in, according to the rules of the ISO standard IS-8601, section 5.5. A week that lies partly in one year and partly in another is assigned a number in the year in which most of its days lie. This means that week 1 of any year is the week that contains 4 January or, equivalently, week 1 of any year is the week that contains the first Thursday in January. Most years have 52 weeks, but years that start on a Thursday and leap years that start on a Wednesday have 53 weeks. January 1 may well be in week 53 of the previous year!
- ***Month of year:*** January to December; note that in GregorianCalendar, January is month 0. In contrast, in DateFormat, January is month 1.
- ***Time stamp:*** An instant in cosmic time expressed in milliseconds since January 1, 1970, 0:00 in UTC. It can be a positive or negative 64-bit-long number. These are sometimes called *dates* or *times*.
- ***Offset:*** How many milliseconds difference local time is from UTC. If you live in North America this will be a negative number. It's the sum of both the zone and DST offsets. You add the DST offset and the zone offset to UTC to get local time.
- ***UTC:*** For all practical purposes it's just standard time in Greenwich, England. The people in Greenwich use daylight saving time, so the term GMT or Greenwich time would be ambiguous.
- ***Week of year:*** There are many possible definitions. The default GregorianCalendar definition depends on whether you consider Sunday or Monday as the first day of the week, setFirstDayOfWeek (the default is locale specific), and how many days you insist must be present in the first week of the year, setMinimalDaysInFirstWeek (default 1). The first week of the year is week 1. January 1 may sometimes be considered in week 53 of the previous year.

CLASS	PURPOSE
java.text.DateFormat	Used to convert a date to or from a String. Contains an associated TimeZone.
java.text.SimpleDateFormat	Used to convert a date to or from a String when you want precise control over the format. Contains an associated TimeZone.
java.util.Calendar	Abstract class that's the mother of all Calendars such as GregorianCalendar. It owns the dozens of magic date constants such as Calendar.JANUARY = 0; Calendar.SUNDAY = 1; and Calendar.YEAR = 1.
java.util.Date	Sun's first attempt at a Date class. I refer to it as the lemon of Java. It's now almost completely deprecated. It's now basically just a wrapper around a UTC date/timestamp long milliseconds since 1970. Unfortunately, it's still not completely gone.
java.util.GregorianCalendar	Used to do date calculations. Each GregorianCalendar contains a UTC timestamp and a TimeZone.
java.util.TimeZone	Contains the name of a time zone and how many hours' difference from UTC that time zone is. It also contains the rules for when daylight saving begins and ends. Time zones are named after cities. They're not the usual names.
cmp.business.BigDate	A simple date class for pure date calculations when you don't want the complication of TimeZones and times. Not part of Sun's libraries. You can download it from http://mindprod.com/products.html#BIGDATE

TABLE 1 Classes to solve problems involving dates

- ***Zone offset:*** How many milliseconds difference local time is from UTC if you ignore any daylight saving time correction. If you live in North America, this will be a negative number. You add both the DST and the zone offsets to UTC to get local time.

The Cast

You need to use quite a few different classes to solve even a simple problem involving dates (see Table 1). For details on these classes download the Sun JavaDoc from http://java.sun.com/j2se/1.3/#documentation.

Displaying a Date

This program converts a date to a String for display using the default date format (see Listing 1). That format depends on what the user has configured as his or her preferred date format in the OS.

Note: You must tell the SimpleDateFormat your GregorianDate twice, once to get the time zone and once to get the time stamp.

Displaying a Date with Precise Control

If you want to control how your date looks, use a mask like the one in Listing 2.

Note: You must tell the SimpleDateFormat your GregorianDate twice, once to get the time zone and once to get the time stamp.

Parsing a Date

To convert a date from a String to an internal format is quite a production. Listing 3 shows one way of doing it.

Note: Be very careful with time zones. If you don't specify one, your date will be interpreted using the the local default time zone. If the user hasn't configured it correctly in his or her OS, you may get Pacific standard time or GMT without warning.

Elapsed Time in Hours Between Two Time Stamps

Look at the example program in Listing 4 to calculate how many hours until the next presidential inauguration.

Note: Specify the time stamp with GregorianCalendar.set in terms of eastern standard time, not UTC. Internally the time stamp is stored as UTC.

- Don't create the TimeZone object with new.
- The getTime().getTime() is not an error. The first getTime retrieves a Date object from GregorianCalendar, and the second retrieves a time stamp from the Date object.
- Elapsed time in days is not simply hours/24. It's much more complicated than that.

How Long Until Christmas, Daddy?

This sounds like a simple problem. Programmers posted many different solutions to the comp.lang.java.programmer newsgroup before the gurus stopped finding holes in the logic. Part of the problem is that the question can have many answers. I show eight plausible solutions in Listing 5.

Gotchas

java.util.GregorianCalendar has far fewer bugs and gotchas than the old java.util.Date class, but it's still no picnic.

Had there been programmers when daylight saving time was first proposed, they would have vetoed it as insane and intractable. With daylight saving there's a fundamental ambiguity. In the fall when you set your clocks back one hour at 2 a.m., there are two different instants in time both called 1:30 a.m. local time. You can tell them apart only if you record whether you intended daylight saving or standard time with the reading. Unfortunately, there's no way to tell GregorianCalendar which one you intended. You must resort to telling it the local time with the dummy UTC time zone to avoid the ambiguity. Programmers usually close their eyes to this problem and just hope nobody does anything during this hour.

Millennium bugs are still not out of the calendar classes. Even in JDK 1.3 there's a 2001 bug. Consider the following code:

```
gc = new GregorianCalendar();
gc.setLenient(false );
/* Bug only manifests if lenient
set false */
gc.set(2001 , 1, 1, 1, 0, 0);
nYear = gc.get (Calendar.YEAR);
/* throws exception */ The bug dis-
appears at 7AM on 2001/01/01 for
MST.
```

GregorianCalendar is controlled by a giant pile of untyped int magic constants. This technique totally destroys any hope of compile-time error checking. For example, to get the month use GregorianCalendar.get(Calendar.MONTH)).

GregorianCalendar has the raw GregorianCalendar.get(Calendar.ZONE_OFFSET) and the daylight saving GregorianCalendar.get(Calendar.DST_OFFSET), but no way to get the actual time zone offset being used. You must get these two separately and add them together.

GregorianCalendar.set(year, month, day, hour, minute) doesn't set the seconds to 0.

DateFormat and GregorianCalendar don't mesh properly. You must specify the Calendar twice, once indirectly as a Date.

If the user hasn't configured the time zone correctly, it'll default quietly to either PST or GMT.

In GregorianCalendar, months are numbered starting at January=0, rather than 1 as everyone else on the planet does. Yet days start at 1, as do days of the week with Sunday=1, Monday=2...Saturday=7. However, DateFormat.parse behaves in the traditional way with January=1.

Learning More

The calendar classes are full of surprises. To learn more about them see http://mindprod.com/gotchas.html#DATE.

Dealing with pure dates is much simpler using the BigDate class. Download it from http://mindprod.com/products.html#BIGDATE.

Author Bio

Roedy Green maintains the Java glossary of Canadian Mind Products (http://mindprod.com). He spends most of his time in the comp.lang.java. newsgroups helping newbies and has been writing custom computer programs since 1963.*

roedy@mindprod.com

Listing 1

```
import java.text.DateFormat;
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.TimeZone;
// display inauguration day as a String
public class Inauguration1
{
static public void main(String [] args)
{
// inauguration day is Thursday 2005 January 20 noon Eastern
  Standard Time.
TimeZone est = TimeZone.getTimeZone ("America/New_York");
GregorianCalendar inauguration = new GregorianCalendar(est);
inauguration.set(2005, Calendar.JANUARY, 20, 12, 0, 0);

// locale specific: e.g. Jan 20, 2005
DateFormat df = DateFormat.getDateInstance();

// set timezone
df.setCalendar(inauguration);

// set timestamp
String dateString = df.format(inauguration.getTime());
System.out.println (dateString);
 }
}
```

Listing 2

```
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.TimeZone;

// display inauguration day as a String
public class Inauguration2
{
static public void main(String [] args)
{
// inauguration day is Thursday 2005 January 20 noon Eastern
  Standard Time.
TimeZone est = TimeZone.getTimeZone ("America/New_York");
GregorianCalendar inauguration = new GregorianCalendar(est);
inauguration.set(2005, Calendar.JANUARY, 20, 12, 0, 0);
// mask for: Thursday 2005/01/20 12:00:00 PM EST Eastern Standard
Time
SimpleDateFormat sdf = new SimpleDateFormat("EEEE yyyy/MM/dd
hh:mm:ss aa zz : zzzzzz");

// set timezone
sdf.setCalendar(inauguration);

// set timestamp
String dateString = sdf.format(inauguration .getTime());
System.out.println (dateString);
 }
}
```

Listing 3

```
import java.text.ParseException;
import java.text.SimpleDateFormat;
import java.util.Calendar;
import java.util.Date;
import java.util.GregorianCalendar;
import java.util.TimeZone;
// parse a date
public class Inauguration3
{
static public void main(String [] args)
{
String dateString = "2005/01/20";
// define the format of the date
SimpleDateFormat sdf = new SimpleDateFormat("yyyy/MM/dd");
// set timezone to use in interpreting the date
TimeZone est = TimeZone.getTimeZone ("America/New_York");
GregorianCalendar inauguration = new GregorianCalendar(est);
sdf.setCalendar(inauguration);
try
{
// set timestamp
Date date = sdf.parse (dateString);
inauguration.setTime(date);
// inauguration is now ready with both timezone and timestamp
}
catch (ParseException e)
{
System.out.println ("bad date");
}
}
}
```

Listing 4

```
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.TimeZone;
// Hours until the next presidential inauguration
public class Inauguration4
{
static final int MILLIS_PER_HOUR = 60 * 60 * 1000;
static public void main(String [] args)
{
// inauguration day is Thursday 2005 January 20 noon Eastern
Standard Time.
TimeZone est = TimeZone.getTimeZone ("America/New_York");
GregorianCalendar inauguration = new GregorianCalendar(est);
inauguration.set(2005, Calendar.JANUARY, 20, 12, 0, 0);
// now is current time, using default timezone
GregorianCalendar now = new GregorianCalendar();
// milliseconds since 1970 Jan 1
long epochInauguration = inauguration.getTime( ).getTime( );
long epochNow = now.getTime().getTime();

double hours = (double) (epochInauguration - epochNow)/
  MILLIS_PER_HOUR;
System.out.println (hours + " hours until the inauguration.");
}
}
```

Listing 5

```
import cmp.business.BigDate;
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.TimeZone;
/**
* How Long until Christmas?
*/
public class Christmas
{
static final int MILLISECONDS_PER_DAY = 24 * 60 * 60 * 1000;
public static void main (String [] args)
{
// How long till Christmas? (Sun's way.)
// this code will give a negative number just after Christmas.
System.out.println ("Your child asks, how long is it until "
+ "Christmas?");
System.out.println ("Here are the possible answers you might give"
+ " to an American child:");
GregorianCalendar now = new GregorianCalendar();
int thisYear = now.get(Calendar .YEAR);

// You may open presents at 7 AM December 25, local time
System.out.println ("You may open your presents at 7 AM Christmas"
+ " morning." );
GregorianCalendar christmas = new GregorianCalendar(thisYear,
Calendar.DECEMBER, 25, 7, 0, 0);
// millis since 1970 Jan 1
long christmasTimeStamp = christmas.getTime( ).getTime( );
long nowTimeStamp = now.getTime() .getTime();
double dayUnitsDiff = (christmasTimeStamp - nowTimeStamp) /
(double) MILLISECONDS_PER_DAY;
System.out.println ("1. It is " + dayUnitsDiff + " day units of
24"
+ " hours until you may open your presents.");
System.out.println ("2. It is " + Math.ceil(dayUnitsDiff) + " day"
+ " units of 24 hours rounded up until you may open your "
+ "presents.");
System.out.println ("3. It is " + Math.round(dayUnitsDiff) +
" day units of 24 hours rounded until you may open your "
+ "presents.");
System.out.println ("4. It is " + Math.floor(dayUnitsDiff) +
" day units of 24 hours rounded down until you may open your"
+ " presents.");

int gmtChristmasOrdinal = (int) (christmasTimeStamp /
MILLISECONDS_PER_DAY);
int gmtNowOrdinal = (int) (nowTimeStamp / MILLISECONDS_PER_DAY);
int gmtDiffInDays = gmtChristmasOrdinal - gmtNowOrdinal;
System.out.println ("5. Children in Greenwich have " +
 gmtDiffInDays + " sleeps (midnight crossings) to go until\n" +
"you may open your presents.\n" +
"They may open theirs even sooner.");

// For children living in the USA, the timezone offset will be
negative.
int christmasZoneOffset = christmas.get(Calendar .ZONE_OFFSET)
  now.get(Calendar .DST_OFFSET);
int localChristmasOrdinal = (int) ((christmasTimeStamp +
  christmasZoneOffset) / MILLISECONDS_PER_DAY);
int nowZoneOffset = now.get(Calendar .ZONE_OFFSET) +
  now.get(Calendar .DST_OFFSET);
int localNowOrdinal = (int) ( (nowTimeStamp + nowZoneOffset) /
  MILLISECONDS_PER_DAY);
int localDiffInDays = localChristmasOrdinal - localNowOrdinal;
System.out.println ("6. You have " + localDiffInDays +
" sleeps (midnight crossings) to go.");

// how long till Christmas? ( with BigDate. )
// this code will give a negative number just after Christmas.
BigDate today = BigDate .localToday();
BigDate nextChristmas = new BigDate(today.getYYYY( ), 12, 25);
System.out.println ("7. Uncle Roedy's Bigdate says it is " +
(nextChristmas .getOrdinal() - today.getOrdinal()) +
" sleeps until Christmas.");
int [ ] age = BigDate .age( today, nextChristmas);
System.out.println ("8. Or put another way " + age[0 ] +
" years and " + age [1] + " months and " + age[2] + " days to"
+ "go.");
}
}
```

Download the Code! www.JavaDevelopersJournal.com

Compuware Corporation's

OptimalJ

WRITTEN BY BOB HENDRY

AUTHOR BIO

Bob Hendry is a Java instructor at the Illinois Institute of Technology. He's the author of Java as a First Language.

bobh@envisionsoft.com

Development of enterprise applications using Java technologies is not for the faint-hearted. Writing to the J2EE specs is proving to be complex, difficult, and tedious – slowing down advanced Java developers and creating a barrier to entry for many mainstream developers.

With advanced Java developers in short supply and even among them, experience with EJB development is rare, thus slowing time-to-market for business applications and challenging application reliability and performance.

To solve this problem we'd ideally want to simplify Java development to allow developers of all levels to build reliable, high-performance components and provide them with a framework for delivering J2EE-compliant business applications. Compuware Corporation believes that it has built such a solution with OptimalJ, a new breed of development environment that enables the rapid design and development of J2EE business applications.

Using OptimalJ, developers can generate complete working applications directly from a visual model, bypassing many of the routine coding tasks associated with EJB development. Design patterns implement best practices for architecture and coding, and an active synchronization feature keeps the code and model in step and up-to-date, allowing application changes at any stage of the development life cycle.

The promise is that OptimalJ, by simplifying the development of J2EE applications, will enable developers of all experience levels to produce reliable applications in a fraction of the time it would take using current development tools.

So how does it do this? Let's step through the features of OptimalJ.

Rapid Enterprise Java Development

Developers use OptimalJ to interact with a visual model of the application that can either be imported from other modeling tools using the XMI or DTD interfaces or built from scratch with the visual modeler. OptimalJ uses this model to generate the architectural submodels and even the working code needed to implement a complete application. Simple graphical windows, editors, and wizards walk developers through each stage of the design, generation, and deployment, reducing the time spent on tedious implementation tasks.

In addition to its model-based interfaces, the integrated development environment based on the open-source IDE, NetBeans, provides a source editor, class browser, form editor, and debugger to enable developers to view, modify, and customize the generated application. So-called "free-blocks" in the generated code allow existing classes to be imported and called by the application, making use of work done outside of OptimalJ.

Using this visual paradigm, developers are shielded from the complexity of coding to the distributed J2EE development. Less experienced Java developers can quickly build or make modifications to business applications. Advanced developers are freed from many of the repetitive coding tasks and can focus on architecture refinements or customization.

Dynamic Business Rules Implementation

Once the basic application structure has been defined in the model, application differentiation can be built-in using a flexible business rule editor. Simple scripting enables developers to add both static and dynamic business rules at the model level. The business rules editor can define referential data constraints, which ensure data integrity and consistency, and event condition rules that provide support for conditional processing. Static rules are generated as Java code in the application, and dynamic rules are stored in a rules database on the application server to allow for modification at runtime. By separating out business rules as easily identifiable elements in this way, many business requirement changes can be quickly and easily implemented.

Pattern-Driven Application Generation

OptimalJ can generate all the application code required for running an application. To do this it first generates models for the Web (JSP), business logic (EJB), and data tiers, which are then used to generate the actual Java code, business rules, and data implementation scripts. The generated models and code are based on implementation templates called *patterns*, which encapsulate knowledge and best practices for coding to the J2EE specification and follow OMG standards. Developers can quickly generate full working applications using JSP, session EJB, and entity EJB technologies with only limited knowledge of the J2EE specs.

Active Synchronization of Models and Code

OptimalJ provides live synchronization between the application models and the Java code. Developers can make changes to applications during or after release using the visual model, and the affected code will be regenerated automatically. The OptimalJ Source Editor identifies managed source code, business rule code, and custom source code to accelerate understanding and enable you to make modifications during development or after release of the software. This feature really underlines one of the core benefits of OptimalJ, one that will become more apparent as Java business applications reach their second and third iterations.

In most projects today the application model is not kept up to date and is discarded once the initial implementation is completed. Developers who join projects late or have to take the first cut and create new versions will benefit from being able to make modifications directly at the model level, confident that OptimalJ has kept the implemented code fully synchronized with that model.

Integrated Deployment Environment

OptimalJ automatically deploys to many of the leading J2EE production servers including the fully integrated Compuware OptimalServer, offered as an option to OptimalJ. OptimalJ also includes an open-source test environment that contains a Web server and EJB container. The OptimalJ deployment packager automatically deploys to this local environment, allowing developers to directly test as they develop without worrying about the complexities of deployment. In the time it would normally take to create the basic business design of an application using a visual modeling tool, OptimalJ can finish coding the entire application and deploy it. Developers can spend their time refining the design and the advanced development staff can concentrate on adding new patterns or customizing the existing ones.

Summary

OptimalJ represents a new paradigm in Java development, promising productivity gains similar to those experienced with the popular and advanced proprietary 4GL products of yesteryear. It adheres to open standards such as J2EE, EJB, JSP, XML, and MOF, uses open source components and industry patterns, and is even IDE and application-server independent. (It supports NetBeans or Forte for Java as its IDE and the iPlanet Application Server, IBM WebSphere, or BEA WebLogic as application servers.)

Compuware has made a clear investment in the J2EE development market and is aligning other products such as its DevPartner and QACenter testing suites alongside OptimalJ to provide a Java tools platform to make development of J2EE applications much easier.

WRITTEN BY JASON BRIGGS

Critical Mass

Technology seems at times to proceed at a breakneck pace. The downside to this expectation for a consistently high rate of technological improvement is that at other times, progress comes at a more leisurely pace – analogous to watching paint dry or sloth racing.

For example, announcements about chip-design developments and the associated higher CPU speeds seem to appear weekly. Intel, IBM, AMD, (name your company), are engaged in a constant competition to outdo the others as they increase speed, reduce power consumption, and introduce new manufacturing processes. However, waiting for those advances to trickle down to consumer devices definitely rates as one of those "paint-drying" experiences.

Another example is the recent announcement regarding organic LEDs (light emitting diodes) by Eastman Kodak and Sanyo. Potentially, organic LEDs will be simpler (and therefore cheaper) to produce than current liquid crystal displays; they'll shine brighter than LCDs and consume less power. A number of large companies are racing to bring these displays to the mass market, and technology like this will eventually have a direct effect on you as a J2ME developer. Forget those stock-ticker and mobile-commerce applications you're working on – it's time for some truly vivid multimedia. Though, on reflection, it will no doubt turn into a waiting game once again before consumers, or developers for that matter, really see the benefit. Might as well go back to that stock ticker after all.

High on my wish list (to see in action), after reading a recent press release from aJile Systems (www.ajile.com), is the JEMcore Java processor core, which uses Java bytecode for its native instruction set. To quote directly from the release: "The aJile hardware, with 100% Java technology software drivers, delivers a 320x240 pixel, 18-bit, full-color display running at 15 frames per second." When you consider that the minimum frame rate for smooth animation is usually cited as 24 frames per second, JEMcore is not quite there, but it's definitely close enough so you can have some serious fun with it.

Also on my list is a Java-enabled PDA from Sharp. Back in March a news item on Bloomberg declared that Sharp would be using Java on its communications devices with PDAs slated to be first. Since then, it has been hard to find any further information on Sharp's plans – so once again it's a case of twiddling your thumbs until a product is officially released.

J2ME presents a unique opportunity for developers. Try as I might I can't think of another time in computing history when a development environment was freely available before most people had access to the hardware. (*Note:* This can mean one of two things: either there hasn't been another occasion, or my knowledge of the history of the industry is embarrassingly lacking.) Whether this is true or not, we as programmers have the unusual chance to become completely familiar with the platform very early on. Certainly you can imagine a critical mass of developers and applications building up in the near future and, as a consequence, even more Java-enabled devices appearing on the market as manufacturers rush to take advantage of this.

In addition to bringing opportunities, J2ME also brings its own novel collection of challenges. The platform is targeted at such a broad range of devices, from the most tiny to things like set-top boxes, which can be close to a computer in size, there's not much chance we'll get bored with the choices. There'll be new virtual machines to come to grips with, new APIs, and new ways of doing things. Micro Edition seems destined to satisfy the saying, "Variety is the spice of life."

It's an exciting time for mobile and embedded Java development. It'll be very interesting to see what happens, both to the platform and within the industry over the next few years.

This is my first foray as an editor for ***JDJ***, and I'm looking forward to bringing you as much useful information about mobile development and the embedded market as can be squeezed into these pages. Perhaps a smaller font and a magnifying glass would be helpful? ✎

jbriggs@abstractminds.com

AUTHOR BIO

Jason Briggs works as a Java analyst programmer in London. He's been officially developing in Java for three years – unofficially for just over four.

JAVA DEVELOPER'S JOURNAL

WRITERS IN THIS ISSUE

BILL BALOGLU, MICHAEL BARBARELLI, JASON BRIGGS, GLEN COATES, GLEN CORDREY, DAVID DEMING, BRADY FLOWERS, NEAL FORD, JEREMY GEELAN, JOHN GOODSON, RICHARD GREEN, ROEDY GREEN, BOB HENDRY, TYLER JEWELL, CHRIS KAMPMEIER, KENT V. KLINNER III, SATYA KOMATINENI, IRVIN J. LUSTIG, JAMES MCGOVERN, JIM MILBERY, PATRICK SEAN NEVILLE, BILLY PALMIERI, AJIT SAGAR, BRUCE SCOTT, DALE B. WALKER, ALAN WILLIAMSON, BLAIR WYMAN

SUBSCRIPTIONS

FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS, PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT

SUBSCRIPTION HOTLINE: SUBSCRIBE@SYS-CON.COM

COVER PRICE: $4.99/ISSUE

DOMESTIC: $69.99/YR. (12 ISSUES)

CANADA/MEXICO: $79.99/YR. OVERSEAS: $99.99/YR.

(U.S. BANKS OR MONEY ORDERS). BACK ISSUES: $10/EA., INTERNATIONAL $15/EA.

EDITORIAL OFFICES

SYS-CON MEDIA 135 CHESTNUT RIDGE RD., MONTVALE, NJ 07645
TELEPHONE: 201 802-3000 FAX: 201 782-9600

JAVA DEVELOPER'S JOURNAL (ISSN#1087-6944) is published monthly (12 times a year) for $69.99 by SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645. Periodicals postage rates are paid at Montvale, NJ 07645 and additional mailing offices. POSTMASTER: Send address changes to: JAVA DEVELOPER'S JOURNAL, SYS-CON Publications, Inc., 135 Chestnut Ridge Road, Montvale, NJ 07645.

WORLDWIDE DISTRIBUTION BY

CURTIS CIRCULATION COMPANY

730 RIVER ROAD, NEW MILFORD NJ 07646-3048 PHONE: 201 634-7400

WRITTEN BY BRUCE SCOTT

The New World of Mobile Computing

By 2004, each corporate knowledge worker will have 3 to 4 different computing and information access devices that will be used to access various applications.

—META Group

In the new world of mobile computing, according to IDC, it's estimated that by the year 2004 there will be 40 million active computing devices, up from 11 million currently. The growth of other handheld devices, like mobile phones, will grow from 300 million devices to a billion by 2004.

What is driving this anticipated growth? There are several factors: the increasing demand for information birthed with the launch of the Internet, and the launch of various technologies enabling the access of information anywhere, anytime, and on everything. One of the primary technologies driving this growth is Java and Java-enabled devices, and the development of Java-based business applications that transform today's gadgets into business tools. The power of Java-enabled devices empowered with a business application using a Java database can always be available and act independently of a constant connection to servers. The user can browse, update offline, and synchronize data occasionally or as required.

Like electricity, information devices will become ubiquitous through every aspect of modern life, providing information where and when it's wanted. The demand for information will push technology to deliver data with low cost of ownership and simplicity. Server-based applications and server appliances will be developed to manage groups of small appliance-like devices. The Java application database will be key to making all of this happen.

We're leaving a world where information is stored in large, complex, and expensive systems controlled by a few and moving to the new world where information is stored everywhere and controlled by many.

Since digital data became available, the demand has been there to free the data from its cumbersome hosts. Many remember that in the beginning data was hosted on expensive mainframes with limited access. Over time, computing started to expand; with the introduction of enterprise servers, desktops, and laptops, data began to free itself more and more. Billions of small points of data are expanding to the outside edge, finding a home within both fixed and mobile Internet appliances, mobilizing information on everything, everywhere, and launching the mobile revolution.

Now, Java applications can extend data beyond servers and onto smart mobile devices such as notebooks and subnotebooks, providing a strong foundation for the deployment of front-office, back-office, and e-business applications to the mobile worker. Moving out and beyond mobile enterprise applications, the unique features of the small-footprint Java application database allow sophisticated applications to be deployed on PDAs, smart phones, and many other Java-enabled consumer devices.

In the new world of mobile data, data is more available and can fit in your pocket and be accessed or updated on diverse mobile platforms. Java is the programming language of the new world. Java aligns with Internet pervasiveness. Java and Java APIs are owned by the developer community, allowing Java to evolve according to the needs of the community, rather than by an oligopoly of a few large technology companies. With Java, a win-win scenario is created. By supporting Java, platform vendors can attract millions of Java application developers. By using Java, application vendors can make their applications available to an unprecedented number of platforms.

For applications on mobile devices to be available always, disconnected usage is the key. Unreliable and costly wireless connections, limited server capacity, and limited battery life make applications dependent on constant wireless access impractical and of limited use. People want their mobile applications to be available regardless of whether they themselves are on an airplane or train or in an elevator shaft. Because of this, the best wireless applications are those that use the wireless connection the least. The wireless connection is used only for periodic bursts of synchronization activity, allowing for the 24/7 access to the wireless application and the data that supports those applications.

What does this mean? It means that knowledge workers will become a part of the most significant technology transformation of the twenty-first century: mobile, pervasive computing. Java will be a core technology enabling the transformation to technology mobility.

bruce.scott@pointbase.com

AUTHOR BIO

Bruce Scott, president, CEO, and founder of PointBase, is a leader in the area of enterprise and embedded database architecture and product development. A cofounder of Oracle in 1977, Bruce cofounded Gupta Technology in 1984, pioneering the notion of the small-footprint database server for Intel-based platforms.

The Incredibly Shrinking Platform

A look at Java for small devices (and other strange tales)

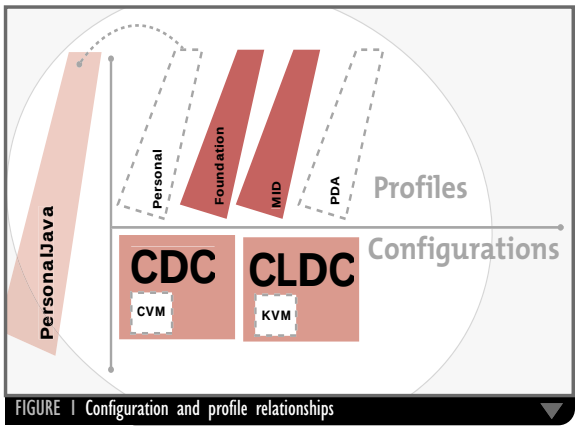


FIGURE 1 Configuration and profile relationships

WRITTEN BY
JASON BRIGGS

Okay. Let's get one thing clear right away. I am not writing (or better yet, dictating) this on a Java-enabled PDA while sitting on the bus on my way to work – a fact that I find somewhat disturbing, and more than a little annoying. Perhaps not as disturbing as my fellow passengers would find it if I was talking into a little box, but disturbing nonetheless. Instead, I'm balancing my notebook on my lap, at a 45-degree angle, while lying on the sofa.

The other way would really be more convenient for those times when I'm just too lazy to sit properly.

Why then, you may ask, is there no cute little coffee cup logo on the bottom right-hand corner of my screen? The answer is that despite the fact that Java 2 Micro Edition software has been out for some time, there are few devices out there for me to play with that have J2ME installed (out of the box). This is not to say that there are *no* devices available, but it's not easy to walk into a consumer electronics store and find something sitting on the shelf, waiting for you.

Impatience aside, it seems likely that relatively soon there will be plenty of devices to choose from. A number of big players in the device market have announced Java support in some form. Motorola jumps immediately to mind, with their iDEN developer community – aimed at providing assistance to embedded Java developers. The expert group for the MID Profile (more on that later) reads like a who's who of big multinationals: AOL, Ericsson, Fujitsu, Matsushita (Panasonic), Mitsubishi, Motorola, NEC, Nokia, NTT DoCoMo, Palm Computing, Sharp, Siemens, Sun Microsystems (obviously), and Symbian, among others, sit in the group. (For the full list, see the MIDP FAQ listed in the References section at the end of this article.)

So what is Java 2 Micro Edition, and why is everyone so keen on it?

The Electronic Coffee Cup

Java (before it was even called Java) was originally destined for consumer electronics devices. Obviously this focus changed along the way, and the Java we know today eventually emerged (for a full history lesson, see the References section). It seems we are now coming full circle and returning to the very devices it was originally intended for. However, despite hardware technology advances, you are unlikely to find a handheld device with the requisite storage space to handle the Standard Edition of Java (a runtime size of around 5Mb) plus the processor requirements to make the most out of it. Which is undoubtedly the reason why PersonalJava emerged, in due course.

More recently Java was split into three main platforms: Java 2 Standard Edition, Enterprise Edition and, of particular interest to us, Micro Edition. In the words of Sun, Micro Edition "specifically addresses the vast consumer space, which covers the range of extremely tiny commodities such as smart cards or a pager all the way up to the set-top box..." (Java 2 Platform, Micro Edition, Sun Microsystems).

This is a shrewd move on Sun's part: the Java space is fragmenting somewhat because of these new platforms. Since an application written specifically for Standard Edition may not necessarily work unchanged on a Micro Edition device, it makes sense to have some form of delineation between the groups. It becomes even more necessary when you look at the number of products that fall beneath the J2ME umbrella.

Configurations and Profiles

Before looking at actual products, it's worth taking a look at how a product is defined within J2ME.

Configurations are the low-level APIs and virtual machine used by a product. There are currently two configurations defined as part of J2ME: the Connected Limited Device Configuration (CLDC) and the Connected Device Configuration (CDC). The CLDC targets devices with 160–512KB of total memory, limited power, possibly intermittent network connectivity, and limited or no user interface. The CDC targets devices with a 32-bit processor, 2MB of total memory and, again, intermittent network connection with limited or no UI.

Profiles sit on top of the configuration and specify all the other APIs required to provide a runtime environment for a particular type of device. The user interface also falls under the control of the profile. Figure 1 shows the relationships between configurations and profiles.

PersonalJava

Most people already know what PersonalJava is, so there's probably not much point in going into detail about the API. Suffice it to say that PersonalJava was there before J2ME emerged as a concept and the specification describes a cut-down version of the Java Standard Edition to fit into resource-constrained devices. PersonalJava, as it stands, does not really fit into the concept of "Configuration and Profile" and hence there are two specifications currently in development aimed at the same kinds of target device – JSR-000075 PDA Profile for J2ME and JSR-000062 J2ME Personal Profile. The latter is "...intended to provide the next generation of Sun's PersonalJava environment, and as such has the explicit requirement of providing compatibility..." with previous versions of the specification (see References). So it looks like PersonalJava will eventually be absorbed fully into the framework.

PersonalJava, in its current incarnation, is probably the most common version of Java you will find on small devices. The

runtime environment from Sun is available for a number of WinCE machines, and compatible software is also available from a number of other manufacturers.

Mobile Information Device Profile

The Mobile Information Device Profile (MIDP) is one of the newer sets of APIs, and targeted at smaller gadgets than PersonalJava. In fact, if you take a look at the MIDP specification, devices are severely constrained. A MIDP device "should" have a screen size of 96x54 pixels, display depth of 1 bit, have 128KB of nonvolatile memory (in other words, memory that is retained when the device is switched off) for the application components, 8KB of nonvolatile memory for persistent data, and 32KB for the Java runtime (taken from the Mobile Information Device Profile JSR-37 JCP Specification, Sun Microsystems, Dec. 2000).

The first home computer I ever used was a Radio Shack TRS-80 (or Trash 80, as it was affectionately known). At the time, this had 64KB of total memory and pixel depth of 1 bit (I don't remember what the screen size was, but I seem to recall the actual "pixels" themselves were about 3mm X 4mm, or something close) – so the MIDP minimum specification is scarily close to the average spec of home computers available in the '70s. Luckily, the machines themselves will be a good deal smaller (I hope, because carrying a mobile phone around that's the size of a Trash-80 would result in one really muscular arm), but the environment you would be developing for is not that far removed.

(*Note:* MIDP sits on top of the Connected Limited Device Configuration [CLDC] and uses the K Virtual Machine.)

Foundation Profile

Another profile currently in development is a device the Foundation says does not require a graphical interface, but "...will also be used as the basis for other profiles that will add GUI as well as other functionality" (from the JSR #000046 J2ME Foundation Profile specification, Sun Microsystems). The Foundation Profile sits on top of the Connected Device Configuration and uses the C Virtual Machine. At the moment it is only available as part of Sun's Community Source Licensing, for Linux or VxWorks, and as such is a little bit beyond the scope of this discussion.

Other Players

There are many other players in the market, supplying virtual machine software for numerous devices. Some have virtual machines that fall within the boundaries of J2ME; others fall somewhere just outside, but are intended for the same market.

IBM has the J9 virtual machine, with runtime support for a variety of architectures (Linux/ARM, Linux/PPC, Linux/x86, QNX in various forms, the Palm OS, and Windows CE). They also have CLDC and a Personal Profile, currently in beta form. Insignia has their Jeode EVM, Wind River has Personal JWorks, Hewlett Packard has ChaiVM, Tao Group has Intent JTE. Most of these are either compatible with PersonalJava or, at the very least, are aimed at a similar type of device.

Support for MIDP is still emerging, but Motorola has a J2ME software development kit, with support for MIDP in their Accompli 008 All-In-One Phone, and the i85s (possibly others as well). NTT DoCoMo supports the KVM on some of their i-mode cellular phones. With Sega developing entertainment content for both, the popularity of the phones would seem assured (see References).

The Great Mobile Land Grab

So success for J2ME appears reasonable, considering wide industry support. But why are all these companies so enthusiastic about the idea?

The answer would appear to lie – at least in part – in the current state of the PC industry. Handheld devices and mobile phones (games consoles, as well) are selling in larger and larger quantities, compared with what seems to be a downturn in the personal computer market. Handheld devices are, of course, cheaper than PCs, and a degree more approachable to the nontechnical public, and fashion is also playing a part in their popularity. The Compaq iPaq is an undeniably sleek piece of hardware (not to forget the Claudia Schiffer Edition Palm PDA although, as much as I squint at the thing, I can't see the resemblance). So with the handheld and mobile market continuing to blossom, Java is, I believe, a logical choice for many of these manufacturers.

The problem with developing software using a language compiled to native code (especially for PDAs and mobiles) is that portability becomes a major issue. The architecture of each platform may differ significantly, and thus, a developer will have to invest considerable time and effort to move their applications from one to the next. Therefore, unless the device has a majority market share, it may be difficult to entice developers to the platform.

J2ME may very well solve this problem for manufacturers. The time to port an application across multiple platforms should be greatly reduced; the developer expends less effort to do so, and the market they're selling to is even larger. The manufacturer gains by having more software available for their product, and the developer gains by the larger market. (I agree that this is an oversimplification of the issues, I merely hope to provide at least one reason why industry support is growing.)

The Master of Many Disciplines

Hopefully, you now have a vague idea of what you might find beneath the branches of the J2ME tree. Wondering what to do with it?

A developer building applications for PersonalJava or MIDP will undoubtedly be expected to be a "Master of Many Disciplines." Applications that must work within a limited environment are likely to rely on a certain amount of server-side support. This will be especially true for MIDP apps (or MIDlets, as they're called). So not only will you need a working knowledge of Standard Edition, but probably of the servlet API, networking using HTTP and/or datagrams, and possibly even J2EE (Enterprise Edition). Putting all those disciplines together, what can you do with them? Pretty much anything.

The first applications to emerge (at least for MIDP) will be entertainment-related. Take a quick look at Bill Day's J2ME archive and you'll find that a majority of the applications offered are games. This is not surprising since games (especially multiplayer) are a good (and fun) way to learn the capabilities of a platform. Game development requires graphics, event handling, thread control, and sometimes networking. My personal belief – probably not shared by the majority, but then it's just an opinion – is that developing games is more challenging, and therefore more rewarding, than developing other types of applications, purely because it exercises your abilities in all those disciplines.

An added advantage is that buyers of these new mobile phones are more likely to download games than other applications, at least until someone comes up with an idea for *the* product that everyone has to have. I've lost count of the number of times I saw people playing "Snake" on Nokia mobile phones when they first came out.

Some Ideas

Multiplayer applets and shockwave apps are a good place to start looking for ideas. Potentially, a basic shockwave app should scale fairly well, down to a MIDlet. For example, looking at something like Habbo Hotel (www.habbohotel.com), which uses simple 3D-styled graphics to create a chat room with a difference, it is

ejip.net

www.vergecorp.com

not unreasonable to think that a similar paradigm would work well even in the restricted MIDP environment. The graphics are sprite-based, and the interface is a simple point-and-click affair (in other words, you point your mouse and click, and your character moves to that position), so network traffic is relatively light. The only really ambitious part of the exercise would be the chat mode. I personally wouldn't want to sit pushing 44-33-555-pause-555-666-0-6-999-0-66-2-6-33-0-444-7777-0-5-2-7777-666-pause-66 on my mobile just to say, "Hello my name is Jason," but who knows, the idea might take off, and as an added bonus, it might create a thriving industry in repetitive-finger-strain-injury medicine.

Games that were popular in the "days of old" will also provide a likely source of material. Don't yawn. Porting a classic game to MIDP might not sound very innovative, or very exciting, but it's another way to learn the technical side of the platform, as well as the limitations, without having to worry about game-play mechanics, balancing, and all the other little issues that go into the design of a game.

Murmurings from Sun

Over the last few months, more and more stories have appeared on the Sun Web site (and on others) regarding Java and gaming. More recently, a Sun press release announced the imminent launch of a new Web site for Java games developers (www.javagaming.org) to provide resource and up-to-date information to the Java gaming community. Hopefully, by the time this article goes to print, the Web site should be up and running.

This support from Sun is another good incentive to get into game development for J2ME. There are very few sites dedicated to game development in Java, so you're forced to glean information from places where there may be as much misinformation as genuine facts and useful resources. An example of this is asking a question at a general game development forum such as, "Has Java3D performance improved enough for games development?" Replies to this could range from outright flame attacks to "I've never tried, but last time I looked it was garbage," instead of something practical like, "I've used it for modeling a walk-through of the architectural design of a new building, and performance was acceptable, but improved when we did this… ."

Decisions, Decisions...

Assuming you're not just experimenting with the technology, and actually want to work on a real product, which of the two main J2ME platforms should you be developing for?

The decision depends very much on your application. PersonalJava provides a much richer API to draw from; the target devices have more screen real estate, faster processors and more memory. MIDP is very limited, which presents its own set of challenges, the devices (likely to be mobile phones initially) have small screens, a minimal amount of memory, and a processor that a snail would find fast, but the rest of us will find somewhat taxing.

Competition may also be a deciding factor in your decision. PersonalJava has been around much longer, so your concept may have been "done to death" on PDAs. Therefore MIDP might be a better choice (assuming that the application will scale to a much smaller device; the next best-selling word processor is not a likely candidate for a mobile phone).

On the other hand MIDlets (MIDP applications) may rely quite heavily on server-side support to provide advanced functionality. Establishing this resource could be a complicated proposition for a small developer. Issues such as middleware and database platforms, initial and projected storage requirements, and scalability – to name just a few – all have to be taken into consideration: setting up database, middleware and Web servers, to support your app, is potentially an exercise in draining your wallet.

Partnerships and development deals might be more accessible for a MIDP application developer than for PersonalJava. With the market so new, the mobile networks will possibly be quite enthusiastic to fill their software catalogs, if only to keep one step ahead of the competition.

C and C++ have an established tradition in the PC and console games market. It has been difficult to argue Java the language, or Java the platform, as a replacement for this tried, tested, and trusted technology. Which is not to say that there isn't an argument for Java on these platforms. However J2ME starts off on a more level footing with other languages, probably on a better footing considering the portability advantages.

One negative thought in all this is that it's not completely clear whether there is definitely money to be made in the burgeoning mobile market for embedded Java developers. The promise is there, but will that promise turn into hard cash? Will all that hard graft actually pay off? These are questions that, certainly when the technology is just emerging, won't be clear until after the fact. I imagine that predicting the direction the industry is going to take would be like trying to foresee the dot-com boom and bust, back before the Netscape team had rolled out their first browser. At the very least, we could expect to see a huge games market emerging for Micro Edition over the next few years.

Footnote

Are you working on MIDP or PersonalJava software, be it a game or a "serious" application? Is your app likely to knock the socks off anything else out there? Or do you have a Java-compatible device that will be the doohickey of choice for the X-Y-Q-R-what-letter-are-we-up-to-generation?

If so, please drop me a line for the possible inclusion of your product in a future article.

References

1. *The J2ME home page:* http://java.sun.com/j2me/
2. *The MIDP Frequently-Asked-Questions List:* http://java.sun.com/products/midp/faq.html
3. *Bill Day's J2ME archive:* http://www.billday.com/j2me
4. *A brief history of Java:* http://ei.cs.vt.edu/~wwwbtb/book/chap1/java_hist.html
5. *The two Community Process specifications that seem to fall within a similar space to PersonalJava:* http://java.sun.com/aboutJava/communityprocess/jsr/jsr_062_pprof.html http://java.sun.com/aboutJava/communityprocess/jsr/jsr_075_pda.html
6. *The Mobile Information Device Profile Community Process specification:* http://java.sun.com/aboutJava/communityprocess/jsr/jsr_037_mid.html
7. *The Foundation Profile:* http://java.sun.com/aboutJava/communityprocess/jsr/jsr_046_j2mefnd.html
8. *Motorola's press release (last November) about Sega developing games for Motorola phones:* www.motorola.com/NSS/Press/press_archive_2000/20001129.html
9. *Information on the Motorola i85s mobile phone:* www.mot.com/LMPS/iDEN/products/i85s/i85s.html
10. *The Accompli 008:* www0.motorola.com/developers/wireless/products/matrix/details/index.html?product=11
11. *Sun's games press release:* www.sun.com/media/pressrel.mar_egaming.html
12. *The new Java Gaming Web site:* www.javagaming.org

Author Bio

Jason Briggs has been officially developing in Java for three years – unofficially for just over four. He works as a Java analyst programmer in London.

jbriggs@abstractminds.com

A UI Framework for the Low-Level MIDP API

I recently had the opportunity to develop Java applications for Nextel's i85s cell phone, the first Java 2 Micro Edition–enabled cell phone in the U.S. In this article I describe some of the issues I encountered when developing the user interface and the framework I created to address those issues. The primary goals of this framework are to allow the development of custom components and to make it easy to assemble those components into screens.

Written by Glen Cordrey

Support for the development of custom visual components

J2ME, CLDC, and MIDP

The Java 2 Platform, Micro Edition (J2ME) includes virtual machine and application program interface (API) specifications for running Java applications in devices with limited computing resources, such as personal digital assistant devices (PDAs) and cellular phones. J2ME defines different configurations that in turn define profiles.

The Connected Limited Device Configuration (CLDC) is intended for devices that have limited (not always on) connectivity to the Internet. The CLDC includes three packages – java.io, java.lang, and java.util – that are subsets of the Java 2 Standard Edition packages of the same name, and the javax.io.microedition package, which is specific to the CLDC. The CLDC also defines the kilobyte virtual machine (KVM), a Java VM that specifically targets devices with limited computing resources. CLDC devices have between 160KB and 512KB of memory available for use by the Java platform.

The CLDC defines a number of profiles; however, the Mobile Information Device Profile (MIDP) is the only one that has been finalized as of the writing of this article. The MIDP targets devices such as cell phones and pagers, with a minimum screen size of 96x54 pixels. MIDP applications are referred to as MIDlets.

The MIDP contains package javax.microedition.rms for record management services, javax.microedition.midlet for MIDlet lifecycle management, and javax.microedition.lcdui containing user interface classes. Within the lcdui package the MIDP distinguishes between what it calls high-level and low-level user interface APIs. Figure 1 shows the primary classes in these APIs.

The High-Level API

In the high-level API, Screen is the base class for screen displays and has four subclasses – Alert, Form, List, and TextBox. Other than a title and a ticker (a string that continuously scrolls across a part of the screen), which are available to all Screen subclasses, List and TextBox each provide only a single visual element on the screen – that is, the structures of List and TextBox are fixed and other components can't be added to them. List displays a list of choices, with each choice consisting of a string and an optional image. TextBox displays and allows the entry and editing of text. The Alert class supports the display of both a string and an image. Form can contain and display any mix of Item objects – one or more ChoiceGroup (containing radio buttons and check boxes), DateField, Gauge, ImageItem, StringItem, and/or TextField objects.

The high-level API targets applications for which portability is essential and relegates look and feel issues to the platforms that the application is deployed on. One price of this portability is that you have minimal control over layout and display attributes such as text fonts and color. In addition, with the high-level API, you can't intercept key events and therefore can't provide contextual filtering of data input.

An Example of High-Level API Limitations

I'll illustrate the limitations of the high-level APIs via the screen shown in Figure 2.

Java Developer's Journal

www.sys-con.com

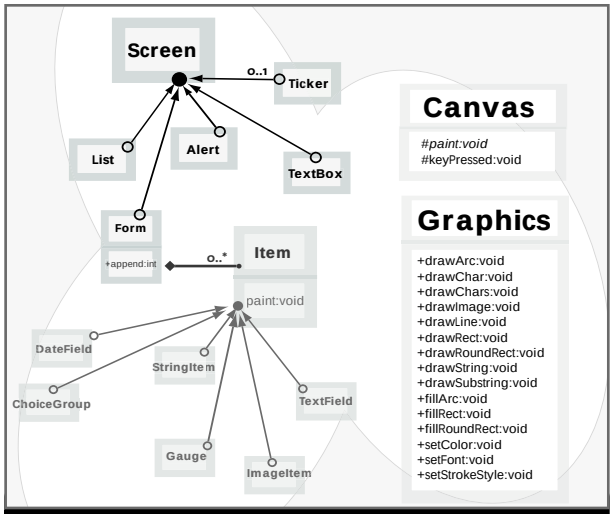


FIGURE 1 The primary MIDP high-level API classes are in white and the low-level API classes in blue

This screen lets users enter expense information into their cellular phone. An expense consists of an amount, an expense type such as airfare or hotel, and the date of the expense. This screen was implemented using the low-level API because the following obstacles would be encountered if implementing with a high-level one:

- ***Layout:*** The amount and date fields are centered on the screen, which is not possible with the high-level API because it doesn't allow you to specify the position of the elements. Also, with the high-level API, the layout for most devices is vertical, meaning that each focusable item (items that receive input) is always placed on a new line. This means that with the high-level API you can't create elements containing multiple data fields on the same line, such as the amount field in Figure 2.
- ***Custom components:*** With the high-level API you can't create custom components such as the expense-type scroll list shown in Figure 2. With the high-level API such a custom component would need to extend Item and then override Item's paint method to paint the custom component on the screen. However, Item's paint method is declared with the package scope, so any class that extends Item needs to be in the javax.microedition.lcdui package.

  Even if you elected to place your custom component in this package, there would be no guarantee that Form's append method, which you would call to add the Item to the display, would treat it correctly, as stated in this response to a message posted to KVM-Interest@Java.Sun.com, Sun's mailing list for J2ME developers: "The high-level Screens like Form don't expose the low-level functions to paint your Item and take events. Most implementations will either not allow you to add a custom Item to a Form or it will only display the Label at best."
- ***Processing key presses:*** While the high-level API's DateField class provides a display similar to the date field in Figure 2, DateField doesn't prevent the entry of invalid dates, such as a month greater than 12, or a day of 31 when the month is 4. Since code that uses the high-level API can't intercept key presses, there's no way for you to add such a filtering capability.

The Low-Level API

The low-level API provides far more control over the look and feel, but at the potential expense of portability (whether or not portability is sacrificed depends upon which specific features of the low-level API are used). In the low-level API, Canvas is the base class for all screen displays and Graphics provides methods for drawing on the screen (although Graphics is also used by the internals of the high-level API; meanwhile, in the high-level API, graphics objects are not accessible to the developer except when using an image). With graphics methods you can draw and fill rectangles and arcs, draw lines, and specify fonts and colors. You can also specify precisely where these elements are drawn on the display. However, the low-level API doesn't contain any predefined visual components such as text boxes or radio buttons. Therefore if you use the low-level API, you have to create your own visual components.

With the low-level API you can also process key events via the keyPressed method in Canvas, which is called by the KVM whenever a key is pressed. You can use this capability to provide the user with the ability to navigate between, enter data into, and select values from your custom components.

The UI Framework

The limitations of the high-level API and the degree of control available in the low-level API drove me to use the latter for my development, and to implement a framework that supports the creation, placement, and display of navigation between, and the processing of key presses by, custom components. A class diagram of the framework is provided in Figure 3. Since these classes are similar in function to some AWT classes, I've followed Swing's lead and given them names that begin with M (for MIDP), followed by the name of the similar AWT class. Also, methods that behave similarly to their AWT counterparts are named the same as in AWT, although the method signatures may differ.

The relationship between MContainer and MComponent is the core of the framework. MContainer (see Listing 1) defines the method add, which adds a component to the container. The code fragment below shows how the ExpenseItem class, which creates the screen in Figure 2, uses this method to add objects of a custom component to the screen display (for brevity I don't show the dollar sign and cents separators being drawn).

```
NumberField dollars =
  new NumberField( 5 );
add(dollars, x, y);
NumberField cents =
  new NumberField( 2 );
add(cents, x +
    dollars.getWidth() +
    3 /* spacer */, y);
```

Figure 4 shows the relationship between these custom classes – ExpenseItem extends MContainer and uses NumberField and MScrollList classes (both of which extend MComponent) for its visual elements.

MComponent and MContainer

In exploring the relationship between MComponent and MContainer let's start by examining the code for MComponent, shown in Listing 2. A component must implement four methods – getHeight, getWidth, paint, and keyPressed. getHeight and getWidth let you position components relative to each other, as shown by the use of getWidth in the previous code fragment. paint and keyPressed will be discussed in detail later. A component also contains its screen coordinates, which are set by the MContainer's add method as seen in Listing 1. In addition, a component contains a reference to the container in which it's placed, which is also set by the container's add method. Finally, a component contains a Boolean indicating whether it has focus.

Web Services Edge

www.sys-con.com

FIGURE 2 Screen for entry of expenses information

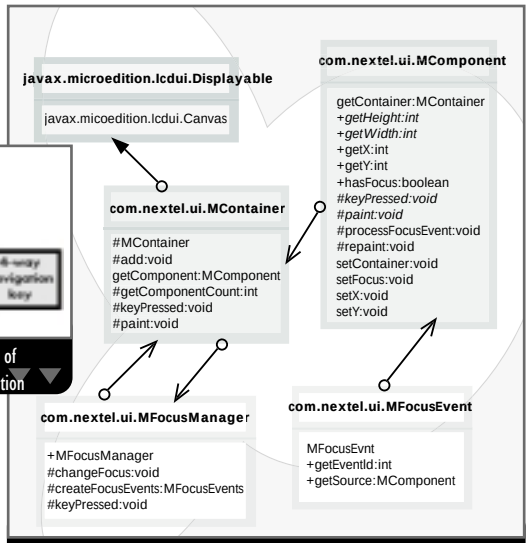


FIGURE 3 Classes in the UI framework

Painting

In the low-level API a Canvas is written to the screen when the KVM calls the Canvas's paint method, which is declared as:

```
protected abstract void paint(Graphics g)
```

The Graphics object passed in this call has methods that draw specific visual elements on the screen. These methods include:

- ***drawArc:*** Draws a curved line
- ***drawChar, drawChars:*** Draws one or more characters
- ***drawImage:*** Draws an image
- ***drawLine:*** Draws a straight line
- ***drawRect:*** Draws a rectangle
- ***drawRoundRect:*** Draws a rectangle with rounded corners
- ***drawString, drawSubstring:*** Draws strings and substrings
- ***fillArc:*** Fills in a circular or elliptical arc
- ***fillRect, fillRoundRect:*** Fills in rectangles with sharp and rounded corners

Graphics also has methods setColor, setFont, and setStrokeStyle (the latter lets you specify whether lines are to be solid or dotted).

Custom components can use the methods in Graphics to provide their desired visual representation. In Listing 2 we see that MComponent declares an abstract paint method, which a subclass implements to display itself on the screen. For example, a class that extends MComponent could display a box containing text by implementing paint as:

```
String text = "Hello World";
protected void paint( Graphics g ) {
  g.setColor( 0x000000 /* black */ );
  g.drawRect( getX(), getY(), getWidth(), getHeight() );
  g.drawString( text, …
```

Since MContainer extends Canvas, in order for components to be painted on the screen, a connection must be made between the container's paint method, which is called by the KVM, and the paint methods of the components within the container. Listing 1 shows how this occurs; when a component is added to the container, it's placed in a vector of components that the container's paint method iterates through, calling each component's paint method in turn.

In addition to painting itself when requested by the container, a component must also be able to initiate a screen refresh after an event that changes the component's appearance, which occurs when characters are entered into a text field as the result of key presses. MContainer inherits from the Canvas method repaint, which can be called. Since repaint is declared with protected scope, a component can call it via the component's container reference (recall that in Java, members with the protected access modifier have both package and protected scope).

In Listing 2 we see that MComponent contains the method repaint, which calls the container's repaint. Typically, after any change that affects the appearance of the component, the component should call its repaint method to ensure that the screen display is updated. As an example, first revisit the code fragment above, which contains a variable named *text*. The component containing this fragment could place digits in the text box by implementing MComponent's abstract keyPressed method as:

```
protected void keyPressed( int keyCode ) {
  char keyChar = (char) keyCode;
  if (Character.isDigit(keyChar)) {
  text += keyChar;//add to existing text
  repaint();
```

Key Presses and Focus Management

The Canvas class provides the only entry point for processing key events; its keyPressed method is called by the KVM to handle keystrokes. Therefore, for custom components to respond to key events the Canvas must pass key events to the custom components.

In Listing 1 we see that MContainer's keyPressed method passes any key press to the focus manager that was created by MContainer's constructor. The function of a focus manager is to support user navigation between fields. I've adopted the convention used in navigating browser forms that moving right (the tab key in browsers) moves to the next component and moving left (shift-tab) moves to the previous component. On the handset shown in Figure 2, the move-right and move-left keys are the right and left sides of the four-way navigation key.

When any key is pressed and the KVM calls the container's keyPressed method, the container in turn passes the key code to the focus manager's keyPressed method. In Listing 3 we see that if the key pressed was the one to move right or left, the focus manager's keyPressed method will, via the changeFocus method, create MFocusEvents (see Listing 4) and call the processFocusEvent methods of both the component that loses focus and the component that gains focus.

In Listing 2 we see that in the component's processFocusEvent method an internal flag is set indicating whether the component has focus. One way this flag may be used by a component is to alter the way the component is displayed when it has focus. For example, via the following code (where "g" is a Graphics object), the component's paint method can display a black background when it has focus and a transparent background when it doesn't:

```
g.setColor( hasFocus()
          ? 0x000000  //black
          : 0xFFFFFF);//clear
g.fillRect( x, y,
     getWidth(),getHeight());
```

The keyPressed method in Listing 3 shows that if the key pressed is neither the move-left nor the move-right key, the key code is passed to the keyPressed method of the component that has focus. MComponent declares this method as abstract, so each subclass must implement code for handling the key press. This returns our discussion to the keyPressed code fragment

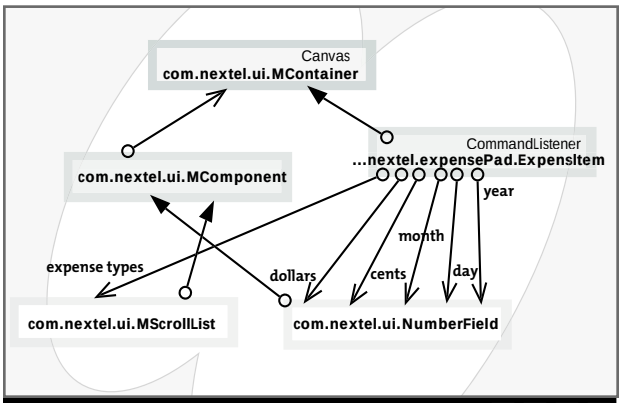


FIGURE 4 The classes that implement the expense item entry screen.

shown earlier, and completes our exploration into how components receive keystrokes.

Enhancements

This framework would benefit from a number of enhancements, including the following:

1. ***Repaint only the affected component:*** Every repaint paints the entire screen, which is inefficient in the typical case in which a component responds to a key press and so only the display of that component needs to change. While I haven't observed any screen flicker from repainting the entire screen each time, it would be more efficient to repaint only the affected area of the display.
2. ***An MPanel class that lets you group and reuse related components:*** For example, with the expense screen shown in Figure 2, I could use one panel to contain the dollars and cents fields and another to contain the day, month, and year fields.
3. ***A layout abstraction so the developer could specify relative positions when adding a component to a container instead of using screen coordinates:*** In AWT and Swing this is accomplished via layout managers. Since some of the functionality that layout managers provide is related to the ability of users to move and resize windows on a desktop display, and because this ability is not currently present on a cellular phone, I suspect that implementing anything other than very simple layout managers for the cellular phone is probably overkill.

Enhancements that address these issues and provide additional functionality, including components such as scroll lists, radio buttons, and check boxes, will be provided in the next generation of this framework, which will be available to registered Nextel developers at http://developer.nextel.com in June.

Summary

The MIDP high-level API provides little developer control over the look and feel and doesn't support the development of custom visual components. The low-level API does, but doesn't provide an AWT-like level of abstraction for using those components. The classes I've described here provide the beginnings of that level of abstraction via the delegation of paint operations from the Canvas to the custom components, and the delegation of keystroke processing from the Canvas to a focus manager and then conditionally to the custom components.

Author Bio

Glen Cordrey is a senior architect at DigitalFocus (www.digitalfocus.com), which develops Internet applications in Herndon, Virginia. He has over 20 years of experience in application development, and has been developing with Java for the last 4 years.

glen@oojava.com

Listing 1: MContainer provides contextual management of MComponents

```
import java.util.*;
import javax.microedition.lcdui.*;
abstract public class MContainer extends Canvas
{
  private Vector components = new Vector();
  private MFocusManager focusManager;

  protected MContainer ()
  {
    focusManager = new MFocusManager( this );
  } // constructor

  protected void add ( MComponent component,
              int x,   // screen
              int y  ) // coordinates
  {
    component.setX( x );
    component.setY( y );
    component.setContainer( this );
    components.addElement( component );
    if ( components.size() == 1 )
    { // first component on the screen
      component.setFocus( true );
    }
  } // add

  // gets component at this rank, where rank is
  // the order in which the component
  // was added to the container
  MComponent getComponent ( int rank )
  {
    Object obj = components.elementAt( rank );
    return ( obj == null ? null :
      (MComponent ) obj );
  } // getComponent

  // gets the number of components in the container
  protected int getComponentCount ()
  { return components.size(); }

  protected void keyPressed( int keyCode )
  { focusManager.keyPressed( keyCode ); }

  protected void paint( Graphics g )
  { // paint each component
    for ( Enumeration enum = components.elements();
   enum.hasMoreElements(); )
    {
      MComponent nextComponent =
 ( MComponent ) enum.nextElement();
      nextComponent.paint( g );
    }
  } // paint

}// MContainer
```

Listing 2: MComponent, the base class for all custom visual components

```
import java.util.*;
import javax.microedition.lcdui.*;

abstract public class MComponent
{
  final MContainer getContainer()
  { return this.container; }

  abstract public int getHeight();
  abstract public int getWidth();

  final public int getX()
  {return this.x;}

  final public int getY()
  {return this.y;}

  public boolean hasFocus( )
  { return this.hasFocus; }

  abstract protected void keyPressed(int keyCode);
  abstract protected void paint( Graphics g );

  // container in which this component is placed
  private MContainer container;
  private boolean hasFocus = false;
  private int x, y; // screen coordinates

  protected void
    processFocusEvent( MFocusEvent event )
  {
    hasFocus =
      ( event.getEventId() ==
        MFocusEvent.FOCUS_GAINED );
  } // processFocusEvent

  protected void repaint ()
  {
    if ( container != null )
    container.repaint();
  } // repaint

  final void setContainer( MContainer container )
  { this.container = container; }

  void setFocus ( boolean state )
  { hasFocus = state; }

  final void setX(int  value)
  {this.x = value;}
```

```
  final void setY(int  value)
  {this.y = value;}
}// MComponent
```

Listing 3: MFocusManager provides navigation between MComponents

```
import javax.microedition.lcdui.*;

public class MFocusManager
{
  // offsets to next and previous component
  private final static int NEXT = 1;
  private final static int PREVIOUS = -1;

  // Values of the keys to move right and left
  private int rightKey, leftKey;

  // Container of components to manage focus for
  private MContainer container;

  // The rank of the component with focus
  private int focusedComponentRank = 0;

  public MFocusManager ( MContainer container )
  {
    this.container = container;
    leftKey = container.getKeyCode(Canvas.LEFT);
    rightKey = container.getKeyCode(Canvas.RIGHT);
  } // constructor

  protected void
    changeFocus( int offset ) // NEXT or PREVIOUS
  {
    // notify currently focused component of change
    MFocusEvent event =
      createFocusEvent( focusedComponentRank,
         MFocusEvent.FOCUS_LOST );
    container.
      getComponent( focusedComponentRank ).
      processFocusEvent( event );

    focusedComponentRank += offset;

    // notify newly focused component of change
    event =
      createFocusEvent( focusedComponentRank,
         MFocusEvent.FOCUS_GAINED );
    container.
      getComponent( focusedComponentRank ).
      processFocusEvent( event );

    container.repaint(); // update display
  } // changeFocus

  protected MFocusEvent
    createFocusEvent( int componentRank,
        int eventId )
  {
    return
      new MFocusEvent( container.
          getComponent(componentRank),
          eventId );
  } // createFocusEvent

  protected void keyPressed( int keyCode )
  {
    if ( keyCode == rightKey &&
         focusedComponentRank <
         container.getComponentCount() - 1 )
    { // not the last component
      changeFocus( NEXT );
    }
    else  if ( keyCode == leftKey &&
        focusedComponentRank != 0 )
    { // not the first component
      changeFocus( PREVIOUS );
    }
    else // pass the key stroke to the component
    {     // with focus
      container.
      getComponent( focusedComponentRank ).
      keyPressed( keyCode );
    }
  } // keyPressed

}// MFocusManager
```

Listing 4: MFocusEvents are delivered to MComponents to notify them when they gain and lose focus.

```
public class MFocusEvent
{
  public final static int FOCUS_GAINED = 1;
  public final static int
    FOCUS_LOST = -FOCUS_GAINED;

  private MComponent source;
  private int eventId; //FOCUS_GAINED or FOCUS_LOST

  MFocusEvent ( MComponent source, int eventId )
  {
    this.source = source;
    this.eventId = eventId;
  } // constructor

  public int getEventId ()
  { return this.eventId; }

  public MComponent getSource ()
  { return this.source; }

}// MFocusEvent
```

Download the Code!
www.JavaDevelopersJournal.com

JDJ Showcase CD

(201) 802-3020

JDJ Showcase CD

(201) 802-3020

Java Thick Clients with J2ME

The pros and cons of implementing Java in restrictive environments

WRITTEN BY
GLENN COATES

This article introduces the development issues relating to thick clients with J2ME and related Java technology such as PersonalJava. It is intended for developers planning to deploy traditional thick client applications on 'consumer devices' such as mobile phones, smart phones, PDAs, and set-top boxes. It also covers material specifically of interest to device manufacturers and Original Equipment Manufacturers (OEMs).

Introduction to Thick Clients

A Recap on Thin Clients

The rather subjective term "thin client" is fairly well-known nowadays; a good example would be a "Web-based application." A thin client doesn't require the developer to design and write the software that runs directly on the client device. Instead, they typically use the features of a browser or some other host application in order to present and manipulate the application data. The real "guts" of the application is actually server-based, which means it can be managed and upgraded centrally.

Note: I have used Web-based applications as an example of thin clients as these are probably the most common.

The thin-client developer generally needs only to develop the server-side software that generates the markup, such as HTML, and maybe some scripts, such as JavaScript, that can be interpreted by a browser in order to define the user interface and perform any local calculations or input validation.

Figures 1 illustrates the main architectural features of a thin client.

As well as reducing client development time, this approach also provides a potentially more efficient deployment mechanism. Anyone with a supported browser can use the application without having to install it in the conventional way. Instead of having to install it via a CD-ROM or downloading via the Internet, the user just enters the URL for the application and performs a refresh if necessary. This can solve many of the expensive and time-consuming upgrade and management problems usually associated with thick clients.

As we can see, a thin-client application does have numerous benefits, though it may have specific disadvantages. In our example of a browser-based thin client, the browser in effect interprets the mark-up language, such as HTML and JavaScript, in order to present the user interface. This means that when it needs to update a page or display a new page from the server it's likely to be slower and less responsive. Many mobile professionals using a device in the field would benefit from an asynchronous communication model due to restricted bandwidth and unreliable wireless connections. This may include allowing the user to do most of his or her work offline, only connecting to the server at key stages in order to "synch" and guarantee that a transaction is processed once and once only. Browsers by their very nature are synchronous, and achieving such behavior would be difficult if not impossible.

An application implemented as a thin client can mean that there is less control over the look and feel; the user interface may seem inconsistent when browsers from different vendors are used. The thin client may have to be developed for the "lowest common denominator" or may have to manage specific features of different browsers. Moreover, there is a risk that it may be susceptible to browser incompatibilities, idiosyncrasies, or bugs. Browsers from different vendors may support different versions of markup and scripting languages.

A thin client also presents a number of restrictions. For example, it is usually not possible to directly access the device's storage area, built-in device applications, or other device resources.

Although it's often stated that a thin client is easier to deploy and upgrade due to its thin Web nature, it does not escape real-world problems. In order to run a thin client, we may have to reconfigure the browser or even reinstall the latest browser release. This may be necessary if a thin client requires the browser to have the up-to-date Java plug-in, allow cookies, or add digital certificates; the list goes on. Many larger companies have strict heavyweight security policies and busy support workloads, and departments may not even have browsers or have predated and different versions. Upgrading these and configur-

ing them for a new thin client is no trivial matter, especially for mobile devices that are out in the field being used by mobile professionals.

Table 1 shows the main advantages and disadvantages of Web-based thin clients.

What Defines a Thick Client?

The rather subjective "thick client" is what many application developers will have already developed on numerous occasions. It is a tier one application designed with a specific task in mind, and is installed directly onto a desktop machine or mobile device. It may use open standards-based protocols such as HTTP or CORBA, but generally it does not use a markup language such as HTML for application presentation. The thick client application alone is usually responsible for presenting the data via a user interface. In contrast to Web-based thin clients, which use a browser to deploy functionality, a thick client must usually provide most of the application features itself. A suitable example is that of a Web browser. The Web browser is very much a thick client, whereas the Web applications that run within the browser can be regarded as thin clients.

Figure 2 illustrates the main architectural features of a thick client.

Why Develop a Thick Client?

A thick client has a number of advantages over a thin client. In most cases the advantages have to do with the higher level of control that can be gained over the device. As we are not relying on the constraints and restrictions of a hosting browser application, we generally have more freedom to develop a user interface embracing the device's full user interface offerings, thus allowing a more "at-home" feeling for already experienced users of the device. Device features such as file system/storage system, built-in applications and other resources, such as infrared or Bluetooth, are usually more accessible if we specifically design the application for the device or device family. This may be very important in the case of a PDA; the built-in applications may be what make the device so useful; failing to integrate with these may limit the usefulness of your application. Additionally, the application can generally be better controlled and made more robust as it is not relying on browser bugs or idiosyncrasies.

It can be argued that, for more frequently used, complex or business critical applications it is usually a better idea to develop a thick client.

Table 2 shows the main advantages and disadvantages of a thick client.

How Would a Java Applet Be Defined — Thick or Thin?

We have now made some reasonable definitions for a thick client in contrast to a thin client, but where would an applet fit into these definitions? An applet is a small Java application that is anchored to an HTML Web page. When we select the link, the applet is downloaded, verified, and run within the browser and the JVM — that is, if our browser is configured to allow this. This is done every time we select the applet link.

Since an applet is Web-based and runs within the constraints of a browser, it is more thin than thick. Another important giveaway is that the user does not have to install it; the applet is downloaded automatically by the browser when the user selects the appropriate HTML link and does not run directly on the device. It is, therefore, reasonable to suggest that an applet is thin. Or, maybe it's a thick thin client!

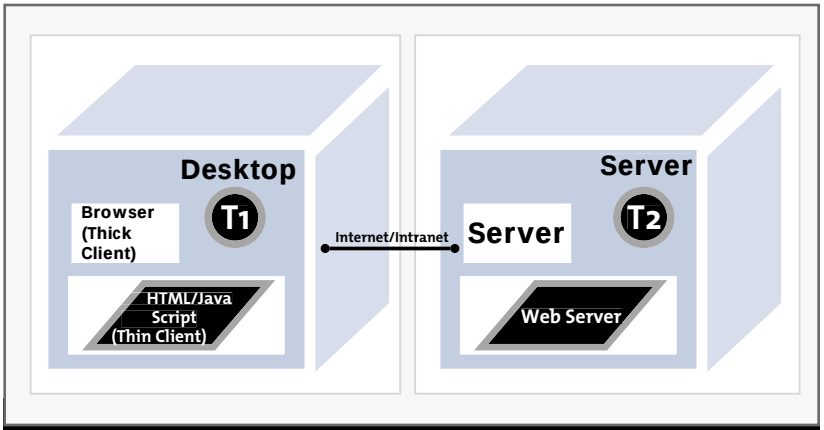


FIGURE 1 Main architectural features of a thin client

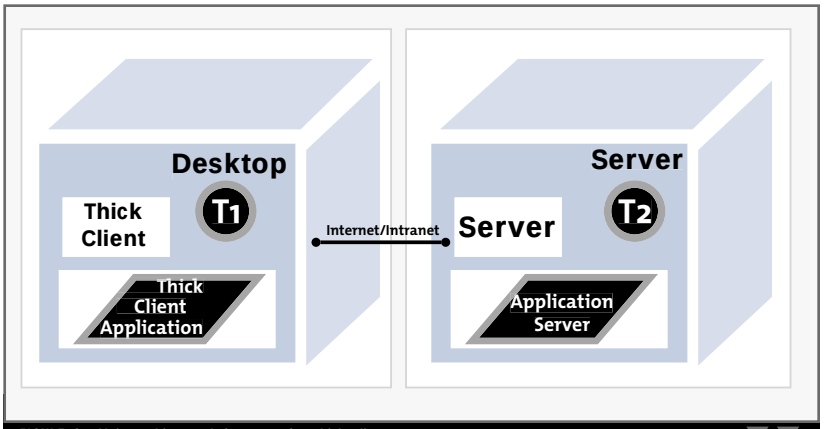


FIGURE 2 Main architectural features of a thick client

Why Use Java for Thick-Client Development?

Productivity and Reduced Risk

Most Java developers who have used other development environments, such as C and C++-based ones, prefer Java for a number of reasons, including the increase in productivity and quality. Although good quality software can be, and is, developed in other languages and environments, it requires more skill and effort, which obviously means higher costs. Java is a "clean" object-oriented language that doesn't inherit the legacy of C and strips away the often misused parts of C++. Commonly required features, such as threading and synchronization, are built into the language itself, and an appropriate garbage collector, when understood properly, usually means fewer memory leaks and is suited to most environments.

Prior to PersonalJava and J2ME, developing software for mobile devices usually involved C or C++ or proprietary languages and toolkits. While some of these toolkits were and still are very productive and appropriate, some were a developer's (and Project Manager's) nightmare that required previous experience and a few "battle scars." Java's predecessor, OAK, was originally designed specifically to address such

WebSphere Journal

www.sys-con.com

problems with consumer device development. Before Java, just attempting to deploy the same thick-client application on a different device imposed many risks, high costs, and delayed time to market. Thus, applications were and still are often targeted at one or two mobile devices; some would argue that this can limit the success of the application and also the device.

Platform Independence

There should be less need to understand the intricacies of the target devices. Once a thick client is J2ME CDLC–compliant it should work well on any device that supports that configuration and has the appropriate Profiles installed and any additional packages. (I'll talk more about J2ME, Configurations, Profiles, and PersonalJava in a follow-up article.) Java provides an abstraction layer between the platform and our application, and because it's ostensibly more open (while under the control of Sun), it helps save us from becoming the victim of the vendor lock-in anti-pattern. If we have problems with a bought-in application or component, it's much easier to exchange it for another. As Java is platform-independent, it means that we will need to "adapt," rather than port, an application for a different mobile device. It means that we can reuse our existing skills without having to learn radically different proprietary development environments just because we want to deploy our application on a new device.

Dynamic Linking

Dynamic linking means that a class need not be loaded and linked until it is actually required. This is ideal for many application environments and allows for more efficient use of resources.

Built-In Security and Integrity

Java does automatic runtime checks such as array bounds checking, null reference checking, and legal cast checking. Arrays are automatically initialized. Synchronization is built into the language. Security management and class verification are also integral.

More Open with Well-Defined Standard Extensions

Along with the core Java editions, numerous standard extensions are

ADVANTAGES	DISADVANTAGES
Centralized administration and configuration and automatic upgrade.	Must run within the constraints and restrictions of the browser. May have to develop the application for the lowest common denominator; or "juggle" differing features. Simpler user Interface. Difficult to "tap into" the device and built-in applications.
Application can be used from any desktop or device that has the appropriately configured host browser.	Application may be designed to run "best" on a particular browser or browser version. This may mean browsers need upgrading or reconfiguring.
Easy to upgrade/update, without the need for a client-side reinstall.	Browsers may offer different levels of support for features such as version of markup language and plug-ins.
Functionality such as security, communications protocol, look and feel taken care of by the host browser.	Browsers may still require re-configuring, for example, digital certificates may need to be installed.
Great for simple or lightweight applications.	Not a generic application – "write once run anywhere." In the real world there are different web-based models such as HTML, WAP and Imode.
Usually cheaper and easier to develop.	

TABLE 1 Advantages and disadvantages of Web-based thin clients

ADVANTAGES	DISADVANTAGES
Maximum control over device resources and extensions.	More development effort required. Longer maintenance cycles.
Full control over user interface.	Takes longer to download and install. Takes more time and effort to upgrade.Will need to be explicitly uninstalled when no longer required by user.
Faster application.	Still have the problem that devices are different by nature with varying user interfaces and input mechanisms. May still need to target specific devices or device families.
Full control over security.	You will need to provide all required features.
Better for asynchronous, connection-less and transaction-oriented applications.	Usually requires more software engineering expertise.
Choice of application-level and communication protocols.	
More control over end quality and reliability.	
Access to device resources and applications.	

TABLE 2 Advantages and disadvantages of a thick client

available. These extensions are defined by Sun with industry input via the Java Community Process. The APIs are well-known and well-documented, with ample examples and tutorials to springboard development. The developer is less open to risk of vendor lock-in and, depending on the extension, can replace an implementation with a better one with minimal impact.

Java Risks

When developing applications with Java, there are a number of common risks that may manifest themselves in embedded or real-time environments. Actually, these are not "Java" risks per se; they are risks associated with JVM implementations and their adoption within restricted environments. Here are the main ones I've come up against:

- Speed
- User interface responsiveness
- Memory requirements
- Memory management — the Garbage Collector
- JVM selection

Speed

Java may be considered "slow" for a number of different reasons. This may be due to Java features such as runtime checks, security checks, dynamic binding, synchronization, and garbage collection. However, another contributing factor here is the way it is traditionally executed, with a JVM interpreter. Don't be fooled; software interpretation is by an order of magnitude slower than machine code executing on the same processor. Many Java implementations rely on this approach to executing the Java byte code, and while it is suitable for many environments, it may not be for others. As we shall see later, most mobile devices will have limited processing power in terms of a lower clock speed in order to reduce battery power consumption. Where the use of a JVM interpreter may provide acceptable performance on the same processor running at a higher clock speed, it may well struggle at a reduced clock speed. So, depending on the device and the application it may be possible to reduce much of this risk if there is the option to use some form of target compilation, for example, JIT compilation or ahead-of-time compilation.

JIT compilers will compile the byte code to the target machine code while the application is being executed. However, there is little point in compiling byte code that is seldom executed, as this will affect the overall performance of the application and introduce "code bloat."

Ahead-of-time compilation involves compiling classes during development. These may be the core Java classes, the application classes, or hot classes picked out by the developer in order to best manage the performance/memory overhead trade-offs. The classes may then be linked with the JVM in order to provide the required performance boost.

Unfortunately, though, while compilation (via JIT, Hotspot, ahead of time) may reduce the speed risk, it may inadvertently introduce other risks such as higher memory overhead, and power consumption and complexity. Compared to RISC and even CISC machine code, Java byte code is very abstract. It is machine-independent and is based on the concept of a stack-based machine interpreter instead of a register-based one (mainly for portability reasons). As such, Java byte-codes can represent more functionality in less space than their machine code equivalents.

Ahead-of-time compilation also introduces another potential problem. It reduces the portability of the application. If we want to deploy an application on a number of devices, we will need to produce separate binaries for each target device. Therefore, unless we are prepared to produce and maintain separate binaries for each target, ahead-of-time compilation may not be suitable. It is for this purpose that the JIT compiler was invented — to provide the benefit of both machine independence and run-time performance!

As memory becomes more compact and less expensive, compilation will probably become more widespread on mobile devices. In addition, JVM chips become much more popular as Java becomes more ubiquitous in embedded and soft real-time systems. Many believe it's the next logical step for JVM chips to "blossom" and become more popular in order to improve performance, increase quality, and reduce costs and power consumption. Shifting the software JVM risk as a hardware concern will soon be the norm for most consumer devices, and is planned as the next generation of JVMs for many device manufacturers. An example of this is the Moon Java processor (www.VulcanMachines.com).

User Interface Responsiveness

This factor is related to the issue of speed discussed above. Of course, the perceived speed of the user interface is of great importance. It has a huge impact on the overall usability and perceived quality of the application. These two issues can be addressed in much the same way as the speed issues. However, it is also worth noting that the application may not need to be lightning fast in order to have a responsive user interface. A device may not need to be fast at all in order to give the user the perception of speed. For example, the user should have immediate feedback from the user interface where possible.

Memory Requirements

Many Java implementations have previously been criticized for being memory-intensive. To be fair, this has usually been due to the JVM or JIT compiler. Remember, we may be running not only our application but a Java byte-code interpreter or a just-in-time compiler as well.

It is also worth noting that if you do use the traditional byte code interpreter in favor of a faster JIT compiler, the memory requirements are likely to be lower. Java byte code is an abstract stack-based intermediate code. As such, it can do a lot more in less space than real target machine code, especially if you're using RISC-based CPUs.

WHY MIGHT A VENDOR PROVIDE ONLY A JVM INTERPRETER?

As previously stated, there is no obvious or best-fit execution method for Java. There are numerous variables and trade offs. However, the reasons why ahead-of-time compilation or JIT compilation may not be available may include the following:

- Developing a JIT/Hotspot compiler for the embedded and real-time world is very difficult. Many mobile devices simply won't be able to afford the extra memory requirements and processing overhead necessary for compilation.
- Vendors usually develop an interpreter first in order to prove the technology and gain experience with JVM development. They then move on to the more sophisticated execution models.
- For some mobile devices, ahead-of-time and JIT technology is just too heavyweight, and a traditional interpreter is more appropriate. For example, if a regular mobile phone is more likely to support very simple applications, then an interpreter is wholly appropriate.

XML Complete Works

www.jdjstore.com

However, there will be a price to pay – speed.

RAM also needs to be set aside for the JVM's internal dynamic data structures, such as the constant pool entries and C stack. The Garbage Collector may also need a large chunk of RAM depending on the internal algorithms used. Most of this is often unnoticed on a desktop machine where ample RAM and virtual memory are available, a luxury that consumer devices don't have.

There may be only a flash-based file system in which to store JAR files, application data, and other resources. The executable JVM or compiled code may be burned into flash ROM and executed directly from flash ROM, while using RAM for stack and heap space only. However, we may only have RAM available; in this case, we have a block of RAM that is to be used for the JVM code, heap and stack space, and also storing JAR files and other resources. Thus, the limited RAM we have is also taking on the burden of a desktop machine's hard drive.

Thankfully, with the increasing memory available in small devices, the memory risks should become less of a problem.

As previously mentioned, the use of Java chips should become a very attractive alternative in order to mitigate the risks associated with software-based virtual machines. Ahead of time, just in time or any other type of compilation is simply not necessary as the Java chip directly executes the Java byte-code; this means that portability is maintained while executing the majority of byte-code similar to how a regular CPU executes machine instructions – fast. This reduces the amount of memory required by obviating the need for interpretation or on-the-fly compilation. It also means that compiled target binaries don't need storing. Because all this "heavyweight" processing is reduced, it usually has the advantageous side effect of reducing power consumption. An example of this technology is the MOON Java chip for deeply embedded Java solutions. See www.VulcanMachines.com.

Memory Management – Garbage Collection

When developing a Java thick-client application for a desktop environment with a powerful high-speed processor, 256MB or more of RAM, virtual memory, and a mass of hard disk space, it may be possible to ignore the implementation details of the garbage collector. The chances are that in this type of environment any idiosyncrasies will go unnoticed; the machine will usually be rebooted at least once a day. With most consumer devices such as smartphones, PDAs and set-top boxes, however, this simply is not an option.

If you are a device manufacturer or OEM then understanding the implementation details of the Garbage Collector is imperative. If you a thick-client developer, then knowing how the garbage collector will behave (and thus affect your program) is also important.

JVM Selection

If you're a device manufacturer or OEM, then you will probably have to decide which JVM implementation to integrate. An increasing number of JVMS are being marketed at the embedded and "soft" real-time industry. However, it is imperative that these are properly evaluated and understood before making the deal. This may seem a somewhat naïve statement, but it's surprising how many decisions are made solely on marketing information and so called "strategic alliances"; hence, usually the marketing and selling of any product is geared toward people who hold the purse strings.

The golden rule here is: be wary of what you are told, no matter which badge it has been stamped with or how glossy the marketing brochures are. JVM slection will be a pivotal decision that will have a bib impact on the success of your project.

Developing for Consumer Devices

So, if we're used to developing Java applications for the desktop environment, what differences are there when developing for consumer devices? Here's a list of the more important ones. The important thing to remember is that we are no longer developing for a desktop or server environment. We are developing for a restricted, embedded and possibly real-time environment.

- Memory availability
- Less processing power
- User Interface restrictions and differing metaphors
- Communication bandwidth/Unreliable Connection
- Battery power
- Higher reliability requirements
- Lack of file system

Memory Availability

As discussed earlier, we know that we have less memory on a wireless device. On a desktop PC we may be used to 256MB or more of RAM. On a mobile device we may have anything from 1/2 MB to 64 MB. This means we need to respect the memory available, be realistic when specifying requirements, and take care when designing, coding, and testing our application. We may have to place more emphasis on profiling, optimizing and garbage collection validation.

In order to help use our RAM wisely, we may be able to filter out unused classes, pre-load classes, and execute the JVM and application directly in flash ROM.

Less Processing Power

Compared to a desktop PC, the processor is likely to run at a much lower clock speed and provide far fewer MIPS, in order to conserve the limited battery power. On the other hand, most users will still compare our application's responsiveness to that found on a desktop machine.

User Interface

The user interface of a mobile device is likely to be a major limitation. The interaction model may be very different from what the developer is accustomed to in a desktop environment. The user interface components available are likely to be as limited as the available screen real estate and resolution and display quality. The interaction metaphor may be very different; other input methods may be used, such as a stylus, numeric telephone keypad or soft keyboard. Thus, a developer will have to have a "mind shift" in designing and developing the user interface for a mobile device and fully understand the look and feel of the device.

Bandwidth/Unreliable Connection

Unlike the thick-client application running on a desktop connected to a high-speed network, we may only have GSM or SMS connectivity. In the case of a paging device, we may be restricted by the number of characters we can send per month and only be able to communicate with the server using asynchronous two-way paging messages. We are also likely to have application protocol restrictions. For example, we may only have sockets or a paging protocol rather than RMI and HTTP. It's normal for the connection to be unreliable; if too many users are on the network our connection may be dropped halfway through a transaction. In the mobile world this is "normal" behavior that needs to be handled as such.

Author Bio

Glenn Coates has been a practicing software engineer for seven years. For the last four years he has worked with mobile devices and Java, developing products such as smart-phones, micro-browsers and digital set-top boxes. Glenn holds a BSc (Hons) degree in computer science and is also a Sun-certified architect for Java Technologies. He works for Vulcan Machines developing J2ME Java chip technology.

Battery Power

A mobile device, due to its very nature, will spend most of its time relying on battery power. For this reason, the clock speed is usually very low. The application developer can do other things in order to preserve power, such as making applications easier to use and allowing information to be retrieved quickly.

Higher Reliability Requirements

A mobile device, obviously, needs to be available when the mobile professional needs it. The user doesn't expect to have to reboot the device if they want to call the emergency services. Due to the limited resources, memory leaks or other memory-related problems are likely to manifest themselves very quickly on a mobile device, drastically affecting its usability and therefore its success.

Lack of File System

Many mobile devices lack a filing system or may rely on a Flash filing system. Flash filing systems have a maximum number of writes and need careful management to maintain reliability. Fortunately, this is usually provided by the Flash driver. Some mobile devices may not have a filing system at all and a database may be used in its place. As a device manufacturer developing a closed mobile device you may have to build the Java thick-client byte-code as part of the JVM and define entry points for your applications. However, if you're in the majority and developing an application for an open device, then either a filing system or an alternative database will usually be available.

Summary

This first article in a series looking at the development issues associated with mobile thick clients and J2ME has started by clarifying the general differences between thick clients and thin clients, while also attempting to define the somewhat subjective terms *thick* and *thin*. I then looked at the great advantages of using Java technology for developing thick clients for mobile and consumer devices, before balancing this with the potential risks that may be associated with particular Java implementations in restrictive environments. Developing for mobile and consumer devices in contrast to desktop and server environments was then highlighted.

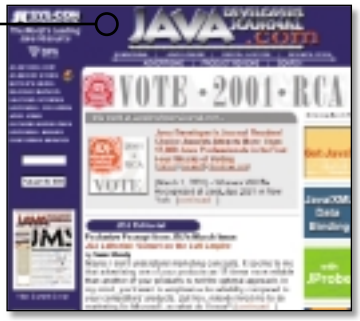
In the next article I'll take a look at Java Editions, in particular J2ME and Personal Java, and also at some of the issues involved in user interface development and architectural design of mobile Java thick clients.

glenn@vulcanmachines.com

What's Online This Month...

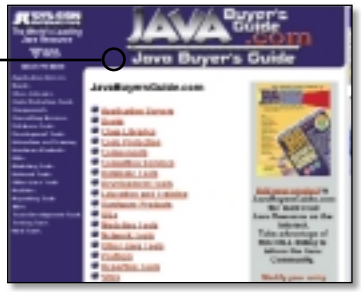
JDJ *Online*

JavaDevelopersJournal.com is your source for industry events and happenings. Interested in Java news and new developments? Check in for up-to-the-minute updates. Can't get to the newsstand? Don't worry, get the info you need online. You can get it from anywhere, anytime.



JavaBuyersGuide.com

JavaBuyersGuide.com is your best source anywhere, anytime on the Web for Java-related software and products in more than 20 mission-critical categories, including application servers, books, codes, IDEs, modeling tools, profilers, and more. Check the Buyer's Guide for the latest and best Java products available today.



Java Jobs

If you're an IT professional and are curious about the job market, you've come to the right place. ***Java Developer's Journal*** offers a portal for IT professionals, giving them direct access to the best companies in the nation.

Simply type in a keyword and a job title along with a location, and get instant results. Search by salary, company, and even industry.

Need more help? Our experts can assist you with retirement planning, putting together a resume, immigration issues, and more.

JDJEdge Conference

JDJEdge 2001 International Java Developer Conference and Expo will take place in New York City, September 23–26, 2001. Sponsored by **SYS-CON Events**, the conference program will present exclusive keynotes, and focus on cutting-edge Java technologies in multiple simultaneous tracks, night school, and an accelerated weekend program.

Look for daily updates, news, and registration information about **JDJEdge 2001**, the largest Java developer conference and expo ever on the East Coast.

JDJList

Join the JDJList, the new Java mailing list community. Join other IT professionals, industry gurus, and ***JDJ*** writers for Java discussions, answers to technical questions, and more. Voice your opinions and assessments on topical issues or hear what others have to say.

Join the JDJList now to monitor the pulse of the Java industry.

Special JDJ *Offers*

Check in for special offers on free seminars, Web conferences, how-tos, and even free software downloads.

JDJ *Readers' Choice Awards*

Vote for your favorite Java software, books, and services in our annual ***JDJ*** Readers' Choice Awards, January 10 through May 30, 2001. Winners will be announced at JavaOne 2001 and honored at the **JDJEdge 2001 International Java Developer Conference and Expo.**

ColdFusion Edge

www.sys-con.com

ColdFusion Edge

www.sys-con.com

XML Edge

www.sys-con.com

XML Edge

www.sys-con.com

Oracle Enhances Oracle9*i*AS

Upping the ante

WRITTEN BY
ALAN WILLIAMSON

The application server battleground continues to heat up. iPlanet has recently announced some major improvements to their product line. Oracle, too, is wrapping up a new release of their Oracle9*i* Application Server (Oracle9*i*AS) product with updates to their caching and Java technologies.

While BEA's WebLogic and IBM's WebSphere are often named as the front-runners in the application server race, we're clearly seeing incredible levels of technical advancement across the board in the middleware space. The Java platform is the technology that makes such rapid improvements possible – and the development community is the beneficiary of all of this largesse.

Oracle and Akamai made news recently when they announced that they've collaborated on a new way to improve the scalability of Web sites and applications. The technology is called Edge Side Includes (ESI) and it defines a simplified markup language that identifies Web-page fragments that can be cached on ESI-enabled servers for faster delivery to end users. Dynamically generated Web pages provide a more personalized browsing experience, but the resulting computational overhead can be overwhelming to a site's infrastructure.

ESI promises to address this issue by enabling the assembly of both cached and noncached page fragments at the edge of the network or the edge of the data center. Edge servers don't normally execute business logic, so they can run on less-expensive hardware and provide superior performance.

Oracle and Akamai have pledged to make ESI an open standard and have invited other vendors to join them in submitting proposals to the appropriate standards bodies. To use ESI, developers embed ESI tags in their Web pages or use JESI (Edge Side Includes for Java) to generate ESI syntax automatically from J2EE applications. Web sites and applications that make use of ESI can be deployed on application servers or content delivery networks that are ESI-enabled.

Both Oracle and Akamai will support ESI in their own products, including the upcoming Oracle9*i* Application Server and Akamai's EdgeSuite content delivery network. For example, an ESI-enabled J2EE application running on Oracle9*i*AS will be able to seamlessly take advantage of Akamai's global network of over 10,000 ESI-enabled content delivery servers (as shown in Figure 1).

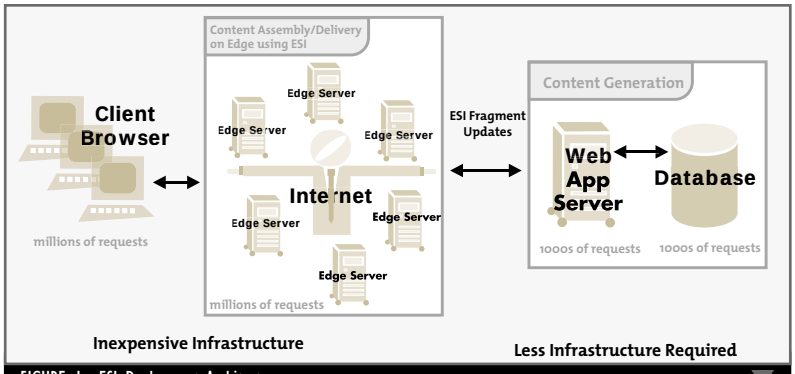


FIGURE 1 ESI Deployment Architecture

The J2EE containers in the upcoming Oracle9*i*AS are rumored to be extremely lightweight and easy to install, and are expected to include enhanced support for EJB 2.0. The product also includes updates for performance and scalability, with new load-balancing and clustering for Java applications. Oracle hinted that there will also be additional session state replication and cluster failover capabilities, as well as Transparent Application Failover (TAF) and database state management.

Every Oracle9*i*AS middle-tier server communicates with an Oracle database using a connection pool, which multiplexes clients on a smaller number of connections to the database, improving performance and scalability. Every application writes its long-lived state persistently to a JDBC-enabled database. When the Oracle database is deployed in a cluster configuration and a specific node in the database fails, all state maintained in the database is transparently "failed-over" to another node in the cluster.

When Oracle9*i*AS is used in combination with the Oracle database, the middle-tier servers have the ability to transparently reroute JDBC connections and database requests to the failed-over node. Depending on how the connections are set up, Oracle9*i*AS promises to provide configurable degrees of TAF with the database – cold, warm, and hot failover of persistent state.

The new Oracle9*i*AS release is slated to be available on a broad range of hardware platforms, scaling from low-end uniprocessor machines to high-end SMP clusters, and on all of the major application server o/s platforms, including Solaris, HP-UX, AIX, Tru64, Windows NT, and Linux. Furthermore, the updated performance for J2EE applications promises to provide "cluster" support independently of the hardware platform or OS being used (i.e., its cluster capability doesn't require a specific SMP hardware configuration). This allows J2EE developers and administrators to be able to leverage these features without being tied to any OS/hardware platform.

These exciting new enhancements will undoubtedly "up the ante" in the hotly contested application server market. Watch for this release from Oracle in the near future.

alan@sys-con.com

Brokat Technologies Achieves Java 2, Enterprise Edition Compatibility

(San Jose, CA) – Brokat Technologies' Brokat Advanced Server/J version 4.6 has been branded J2EE compatible for the Java 2 Platform, Enterprise Edition 1.2 specification from Sun Microsystems, Inc. J2EE compatibility strengthens Brokat Technologies' portfolio for financial institutions, telecommunications companies, enterprises, portals and e-businesses, everywhere, to be able to leverage a standard compliant approach to user-centric application development.
www.brokat.com.

Flashline Component Manager Offers Convenient and Scalable Web Portal

(Cleveland, OH) – Flashline released Flashline Component Manager, Enterprise Edition (CMEE), which enables, promotes, and measures software reuse. CMEE's new Web-based architecture allows organizations to easily scale reuse initiatives to a broad group of users throughout a distributed enterprise. A Web interface provides the convenience and accessibility of a centralized, portal environment for universal component-based development and reuse efforts.
www.flashline.com

Major Development in Java Testing Achieved with Parasoft's JContract and JTest 4.0

(Monrovia, CA) – ParaSoft announced the release of Jtest 4.0 and a new Java development tool, Jcontract. The new version of Jtest automates unit level functionality using the Design by Contract (DbC) format specification information included in the class. Jcontract, a Jtest add-on or stand-alone tool, can verify system-level functionality and identify class/component misuse by checking DbC contracts at runtime.
www.parasoft.com

Product News from Sitraka

(Toronto, ON) – Sitraka has announced JProbe 3.0, a leading performance tuning toolkit for Java (now with improved usability), JClass 5.0, (now with 3D charting), and DeployDirector 2.0 (now fully JNLP compliant).
www.sitraka.com

Rational Test RealTime Announced at Embedded Systems Conference

(San Francisco,.CA) – Rational Software introduced Rational Test RealTime, a product that automates testing of embedded, real-time, and distributed applications. It also introduced Rational QualityArchitect for RealTime.

Rational Test RealTime manages the tedious and error-prone aspects of implementing, deploying, and running an embedded test environment.

Rational QualityArchitect for RealTime is integrated with Rational Test RealTime to add test automation capabilities to Rational Rose RealTime.

Rational has also added support for the Sun Java 2, Micro Edition platform and Mobile Information Device Profile to Rational Rose RealTime.
www.rational.com.

CocoBase Enterprise O/R Provides EJB Database Accesss for BEA Weblogic Server 6.0

(San Fransisco, CA) – Thought Inc., has announced the integration of CocoBase Enterprise O/Rs with the BEA WebLogic Server 6.0 from BEA Systems Inc. CocoBase simplifies the process of mapping relational data with EJBs for use with BEA WebLogic.

CocoBase boosts developer productivity by instantly mapping relational data, generating entity beans and JSPs, deploying to the BEA WebLogic Server 6.0, and by eliminating the need to hand-code database access for EJB entity beans and JSPs, which can decrease development time by as much as 85%.
www.beasys.com

PointBase's Java Technology-based Database Receives JDBC Certification

(*Mountain View, CA*) – PointBase announced that their database is certified through Sun Microsystems' new third-party Java Database Connectivity (JDBC) technology certification program.

Certification means Sun has validated the PointBase JDBC technology–based driver for use with J2EE compatible products. PointBase had to pass a suite of over 2000 tests in order to receive this certification.
www.pointbase.com

WebGain Launches New Products

(*Santa Clara, CA*) – WebGain has introduced a number of products: Studio Professional 4.5, an integrated development suite that streamlines the process of designing, developing, deploying and improving Java and XML e-business applications; WebGain Application Composer, a visual authoring and integration environment that simplifies enterprise application assembly from existing components, legacy systems, and Web services; and Business Designer 1.0, a visual design and collaboration product.
www.webgain.com

TogetherSoft Enables Together ControlCenter Solution for IBM WebSphere App Server

(*Raleigh, NC*) – TogetherSoft Corporation has announced the availability of Together ControlCenter for IBM's WebSphere Application Server, allowing users to hot deploy and remotely debug EJBs and servlets to a WebSphere Application Server.
www.ibm.com/websphere
www.togethercommunity.com

TogetherSoft Corp Announces J2EE Compatibility

(*Raleigh, NC*) – Together ControlCenter 5 has received J2EE compatibility status from Sun Microsystems. The product's integrated development platform uses both the J2EE Patterns from Sun Microsystems and LiveSource from TogetherSoft to automate the deployment of EJBs, EARs, JARs, and WARs. It also allows clients to extend the TogetherAPI so they can create their own building blocks for deployment to other application servers. Together ControlCenter is the first integrated development platform to earn this status.
www.togethercommunity.com

The JDJ Salary Survey

The results are in

WRITTEN BY
Bill Baloglu &
Billy Palmieri

More than 1,150 readers worldwide responded to the recent *Java Developer's Journal* Salary Survey. The respondents covered a broad spectrum of IT titles and levels of expertise from entry-level to executive positions.

They shared information including city, state, country, title, current salary, previous salary, years of professional experience, years in current position, and years with current company.

As you may already know, the state of the employment market for Java developers is one of our favorite subjects, which is why we write a monthly column for ***JDJ***.

From this wealth of raw survey data that filled a 34-page spreadsheet, we set out to draw some meaningful conclusions about who's making what and where. We focused our attention on the vast majority of responses – the 958 people who live and work in the United States.

That's not to say we're not interested in the many skilled IT professionals working in the international market. But figuring all the rates of exchange for a dozen different currencies would have been a bit much for the already daunting task that lay ahead of us.

Were we able to draw some significant conclusions about the salaries in the IT and developer industries in the United States? We think so. However, when considering the conclusions that we reached, keep in mind we're looking at a very broad range of respondents working in a wide range of companies.

For example, the salaries for respondents who have 1–5 years' experience working in the East Coast/New England region range from $32K to $350K a year. The average salary for 1–5 years' experience in that area is $78K (which does sound a little more normal). But the spread between the lows and the highs can be pretty wild.

Part of this is because respondents range from just-out-of-school Web developers to CTOs. We're also assuming, with just cause, that many of these people are working at high hourly contract rates while others are reporting full-time salaries at permanent positions.

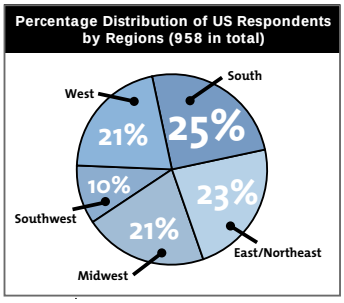
Are we being forced to compare apples to oranges here? Well, yes and no. We're looking at highly paid hourly consultants next to salaried employees, but contract or full-time, the bottom line is: these are the dollars that companies are paying developers to do this kind of work in various parts of the United States.

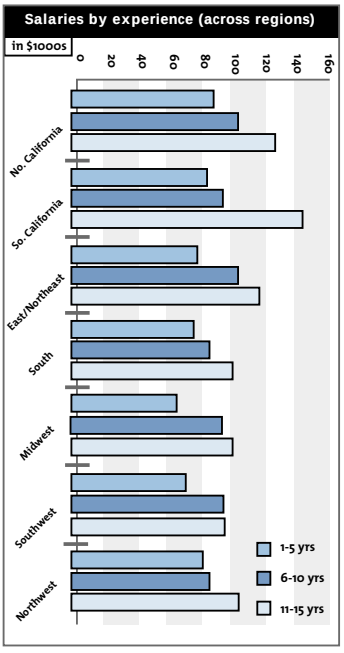
The Big Picture

The average respondent has a total of eight years of professional experience, three years with their current company but only two in their current position. Assuming that respondents are working with Java in their current position, on a national average they haven't been using it for very long.

Eighty-four percent of the U.S. respondents have been using Java for two years or less, and 63% of them have been using it for one year or less. Only 16% of the respondents have been working with Java for more than two years.

Granted, Java has only been around for about six years, but the IT professionals who work with it have leapt to high-paying positions in an incredibly short period of time.





The old tradition of a young apprentice putting in years of training to learn a trade is now a relic of the distant past. The role of the seasoned veteran who trains that young apprentice has been transformed as well, but we'll get to that later.

If you have any doubt that this is an extremely fluid industry with an incredible amount of turnover, consider these statistics. Seventy-seven percent of IT professionals who responded to the survey have been with their current company for three years or less. Sixty-five percent of them have been with their current employer for two years or less, and nearly half of them (46%) have been with their current employer for one year or less.

Part of this could be due to the fact that many of the companies that employ these developers are only 1 or 2 years old. And we all know that turnover can be high even with established companies that are constantly restructuring and downsizing.

The results of this survey clearly support what we have casually observed over the past several years. Company loyalty in the IT industry is close to nonexistent. However, that lack of loyalty is clearly a two-way street.

IT professionals with Java skills hop from one job to the next (84% of them stay for less than two years) for a very good reason – they can. And they know they'll get more money at the next position.

The employing companies contribute to this job-hopping phenomenon by constantly upping the salary ante. Consider this: the average U.S. respondent has been with his or her current employer for three years and is at a current salary of $87K.

That average respondent was making $71K in his or her previous position. This means that across the United States, IT/Java professionals are making an average of 24% more in their current position than they did in their previous position three years ago.

Is job hopping here to stay in this industry? As long as companies continue to pay 24% salary increases to employees (88% of whom won't stay with the company for more than two years), who needs loyalty? In this environment companies are incentivizing IT professionals to keep moving on.

AVERAGES FOR ENTIRE US	
Years in the profession	8
Years in the current position	2
Years with the current company	3
Current salary	$87K
Last salary	$71K

JAVA EXPERIENCE (CURRENT POSITION)	
Duration	Cumulative %
> 2 YEARS	16%
≤ 2 YEARS	84%
≤ 1 YEARS	63%
≤ 10 MONTHS	13%

YEARS WITH EMPLOYER (CURRENT COMPANY)	
Duration	Cumulative %
> 3 YEARS	23%
≤ 3 YEARS	77%
≤ 2 YEARS	65%
≤ 1 YEAR	46%

SALARIES BY REGION

Northern Cal (119 responded)

Years	Average (in $1K)	Range (in $1K)
1-5	85	40-170
6-10	103	50-185
11-15	126	75-250
16+	111	70-200

Southern Cal (59 responded)

Years	Average (in $1K)	Range (in $1K)
1-5	84	55-165
6-10	93	70-140
11-15	144	100-220
16+	104	78-130

East Coast/ New England (217 responded)

Years	Average (in $1K)	Range (in $1K)
1-5	78	32-350
6-10	104	49-300
11-15	116	58-300
16+	115	60-280

South (247 responded)

Years	Average (in $1K)	Range (in $1K)
1-5	75	25-195
6-10	85	40-150
11-15	100	59-207
16+	119	65-275

Midwest (197 responded)

Years	Average (in $1K)	Range (in $1K)
1-5	64	28-112
6-10	94	53-210
11-15	101	65-190
16+	84	62-120

Southwest (92 Responded)

Years	Average (in $1K)	Range (in $1K)
1-5	70	30-120
6-10	94	65-250
11-15	95	60-125
16+	110	64-212

Northwest (27 responded)

Years	Average (in $1K)	Range (in $1K)
1-5	80	60-118
6-10	85	53-114
11-15	104	90-120
16+	92	87-95

The Big Country

The U.S. respondents to this survey were fairly evenly spread across the major regions of the country. Twenty-three percent were from the East Coast/New England area, 21% from the Midwest, 25% from the South, 21% from the West Coast, and 10% from the Southwest.

For salary-survey purposes we divided the West Coast into three regions that are distinctly different markets: Southern California, Northern California/Silicon Valley, and the Northwest.

To get a sense of how IT salaries progress over time, we've categorized respondents by years of experience (1–5 years, 6–10, 11–15, and 16 or more years).

Across the board the vast majority of respondents were in the 1–5 years of experience category, with a huge range of salaries within that category.

This isn't too surprising if you keep in mind that Java hasn't been widely used for much more than five years, and the industry has seen a huge swell in numbers (and demand) within these past four to five years.

The Wild Wild West

In Northern California where this industry was born, the average salary is $85K for 1–5 years of experience, $103K for 6–10, and $126K for 11–15 years. While we expected salaries at this scene of the high-tech gold rush to be the highest in the country, these averages are only slightly higher than those in Southern California.

Southern California respondents earn an average of $84K for 1–5 years, $93K for 6–10, and a healthy $144K for 11–15 years of experience.

Northwest respondents earn a more conservative $80K for 1–5 years, $85K for 6–10, and $104K for 11–15. You might extrapolate that some of them learn the ropes from Mr. Gates, then head south for more sunshine and bigger bucks (but that's merely idle conjecture).

One interesting phenomenon is that in every region we examined (with the exception of the South and Southwest), salary averages took a dip after 15 years of experience. There are a variety of reasons why this might happen.

The industry's economy might be indicating here that after 15 years, IT salaries simply top off. Despite what dot-commers were telling us mere months ago, the sky is not, in fact, the limit.

Industry professionals who've been working in IT for more than 16 years have undoubtedly come up through many different types of technologies, many of which are no longer used. Yes, the mainframe and COBOL experts had a heyday during the Y2K era, but it's a Java, XML, and wireless world now.

It's highly likely that many of these senior engineers are voluntarily taking a decrease in pay as they come up to speed on more current technologies.

But while the young guns of IT rule the West (and most of the country), seniority is still respected in the South and Southwest, which comprise 35% of the survey respondents.

IT salaries follow a more traditional curve in the South, rising from $75K for 1–5 years to $85K for 6–10, $100K for 11–15, and $119K for 16 years or more. In the Southwest the curve grows from $70K for 1–5, $94K for 6–10, $95K for 11–15, and $110K for 16 or more years.

Regional differences may also be due to the different types of companies that employ IT professionals. While the bulk of West Coast companies are in the business of creating new technologies and fiercely competing for hot-shot talent, IT professionals in other regions are likely to be employed by traditional corporations with more conservative salary structures and ranges.

The Personal Picture

As an IT professional, what should I do with this information? If my salary is on the low end of the scale for my region and years of experience, should I storm into my manager's office with this article in hand and demand a raise?

We don't advise it. Again, this is a very broad picture we're looking at that takes only a few key facts about each respondent into account. We haven't looked at the résumés of all 958 respondents and, frankly, we'd rather not.

There are also tremendous variations between these markets and extreme differences in the cost of living in these regions. The IT professional making $75K in the South is most likely living in a much bigger house than his or her counterpart in Silicon Valley.

If you'd like a more detailed comparison of salary ranges for the same type of work in different parts of the country, there are many good resources available, including www.salary.com, which does in fact compare apples to apples.

If you know of other resources that help IT professionals get a good comparative picture of salary expectations nationwide, we'd like to hear about those as well.

We've provided a bird's-eye view of who's making what and where. Will these numbers change dramatically by next year's survey? We don't doubt it at all.

Author Bios

Bill Baloglu is a principal at Object Focus (www.ObjectFocus.com), a Java staffing firm in Silicon Valley. Prior to that he was a software engineer for 16 years. Bill has extensive OO experience ,and has held software development and senior technical management positions at several Silicon Valley firms.

billb@objectfocus.com

Billy Palmieri is a seasoned staffing industry executive and a principal of ObjectFocus. Before that he was at Renaissance Worldwide, a multimillion-dollar global IT consulting firm where he held several senior management positions in the firm's Silicon Valley operations.

billp@objectfocus.com

Cubist Threads

WRITTEN BY
BLAIR WYMAN

Opportunities lost?

I don't follow the press much. In my job, an EDUPAGE subscription is about as committed as I normally get, along with slashdot, the DNRC, and a few other regulars. I see this as a personal failing on my part but never worry too much about it; it's just one of a large list of personal failings.

If I were going to worry about a personal failing, it would probably be my apparent and utter incapability to become a legendary rock star. I even had a sort of a shot at it once, a long time ago.

When I was in my early twenties, I fancied myself a sort of folk musician. I had learned a little guitar here and there, and could carry a tune if the basket was big enough. Well, the late, great Roy Orbison was between major gigs and happened to be playing at a local hometown nightspot. As part of the show that night the band arranged an "open mike" competition, where audience members could come up and sing tunes with the band.

Now, I was actually moving out of town the next day – I had my plane tickets in my pocket and was off to see the world – but I figured I'd take my guitar to the open mike and just see what would happen. It was one of those "loudest applause wins the prize" competitions and I didn't know a soul in the room, but I figured a rousing rendition of something-or-other would win them over.

As Roy and his band watched silently, I sat up there with my guitar and sang a couple of Beatles tunes I knew pretty well at the time. Afterwards I waited for the crowd to erupt in an unrestrained cacophony of thunderous applause and spontaneous universal acclamation.

I waited, and waited, and waited; just as I was thinking of how best to tragically and fatally impale myself on the mike stand, Roy Orbison walked up and gently pulled me away from the spotlight to the side of the stage.

"I really enjoyed your music," he said. "Are you going to be around tomorrow? Maybe we could get together, if that's alright, and play some tunes?"

"Actually, I'm moving out of town tomorrow," I said to him. "Sorry."

Mr. Orbison looked a little disappointed, shook my hand, and wished me luck. What might have happened had I changed my plans and stuck around? Nobody knows, but instead of some obscure JNI guru, I might've become a "Traveling Wilbury"!

So, what am I doing here if I'm such a failure? Am I some sort of Java guru? Definitely not: I don't program much actual Java code in my day-to-day work. However, Java remains at the absolute core of my daily labor. How can I be "doing Java" if I'm not writing it? Simple, really. I mostly work on the JVM; the IBM eServer iSeries JVM, formerly known as the AS/400 Developer's Kit for Java JVM.

So, how did I get here? Why do I get this chance to share my innermost while loops and curly braces?

The answer is simply, "Alan Williamson."

I had the pleasure of meeting Alan for the first time at the Java Migration Conference sponsored by IBM Vienna. I don't get to travel all that much – Vienna is a truly exotic locale for a "Midwesterner" like me – so when someone speaking (something like) English introduced himself to me at the catered lunch, I couldn't help but pay attention. Alan introduced himself as being from a company called "N-ary."

Now, I've done some parameter parsing code so I asked him, "N-ary, eh? Is that like 'unary, binary, ternary,' etc.?"

He said, "Spot on!" So I looked at my tie.

I couldn't see anything on my tie or on my shirt, so I looked up quizzically. It was then that I realized "spot on" must mean something like "yes."

To this day, I think that might have been the last time I fully understood something Alan said to me.

So (near as I can tell), Alan has asked me to contribute to ***JDJ*** from time to time; either that, or to attend a ritual festival of log-throwing, haggis-munching kilt wearers. I'm not sure. I guess if I see this in print I'll know.

AUTHOR BIO
Blair Wyman is a software engineer working for IBM in Rochester, Minnesota, home of the IBM iSeries.

blair@blairwyman.com