BEA WEBLOGIC PLATFORM FREE CD INSIDE

JAVA DEVELOPER'S JOURNAL

The World's Leading Java Resource

March 2002 Volume: 7 Issue: 3

JAVADEVELOPERSJOURNAL.COM

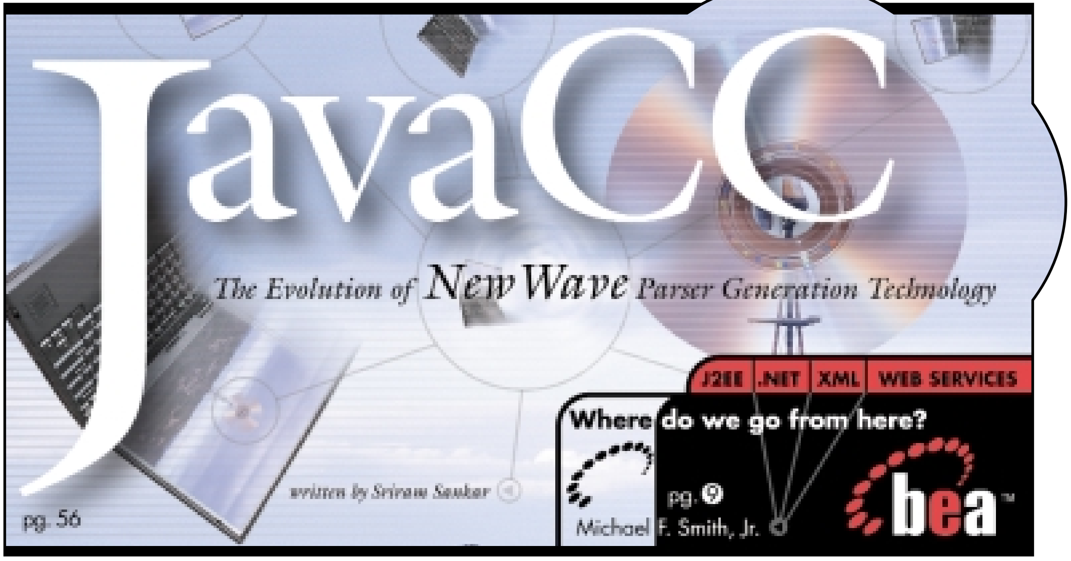
hp application server
an hp netaction product

some of the best things in life just got better

invent

PLUS early access release of hp web services products

www.hpmiddleware.com/download


JavaCC
The Evolution of New Wave Parser Generation Technology
written by Sriram Sankar
pg. 56
J2EE .NET XML WEB SERVICES
Where do we go from here?
pg. 9
Michael F. Smith, Jr.
bea

From the Editor
Alan Williamson pg. 7

JDJ Labs
FrontierSuite 2.2.02 pg. 130
eXtend Workbench pg. 134
Oracle9*i* JDeveloper pg. 140
Book Review pg. 146

Feature: Data Mapping to JDBC, XML & Beyond — *Abstracting Java's many data source/data sink APIs* — *Todd Lauinger* 18

Integrated Systems: J2EE As the Platform for EAI — *Supporting open-standard technologies* — *Matjaz B. Juric* 26

Feature: What Can JCA Adapters Do? — *Are they the right solution for you?* — *David Li* 34

Transaction Systems: End-to-End Transactionality — *Myth or reality? Can modern transaction systems provide guarantees?* — *Mark C. Little* 42

Q&A: Prescriptions for Your Java Ailments — *The answers to all your Java-related questions* — *James McGovern* 66

Feature: An API Developer's Primer — *What's hot and what's not and how to avoid pitfalls* — *Richard Deadman* 72

Java Utility: Extending the J2SE 1.4 Logging Package — *Monitoring made easy with java.util.logging* — *Jim Mangione* 82

Feature: Mac OS X & Java — *A revolutionary step toward usability* — *Hitesh Seth* 88

Jini & J2ME: Jini Surrogate As a Platform for J2ME Games — *Surrogate architecture incorporates smaller devices* — *William Swaney* 100

Feature: Building End-to-End Palm Applications — *A worthwhile and useful experience* — *JP Morgenthal* 106

Mobile Provisioning: Leveling the Playing Field — *A J2ME co-op could let the little guy come out of the cold* — *Jason R. Briggs* 112

Feature: Targeting GPS — *Integrating J2ME, GPS, and the wireless Web* — *Shane Isbell* 118

SYS-CON MEDIA

sonic
software

www.sonicsoftware.com

zero g

www.zerog.com

bea

developer.bea.com

bea

developer.bea.com

togethersoft corporation

www.togethersoft.com/1/jd.jsp

SUBSCRIPTIONS:
FOR SUBSCRIPTIONS AND REQUESTS FOR BULK ORDERS, PLEASE SEND YOUR LETTERS TO SUBSCRIPTION DEPARTMENT
SUBSCRIPTION HOTLINE: SUBSCRIBE@SYS-CON.COM
COVER PRICE: $5.99/ISSUE
DOMESTIC: $49.99/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 $49.99 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.

ALAN WILLIAMSON EDITOR-IN-CHIEF

Java 1.4 and the Rest!

Guess what? It's JavaOne month. That came around fast – and don't panic if you're sitting there wondering where the year went; it hasn't, well not yet anyway. JavaOne is earlier this year, with only a nine-month gap from the previous one. It's moved back to its old March time slot, which makes traveling and staying in San Francisco a little cheaper. Early reports indicate that attendance will be down this year, but I think that's a good thing – it was getting too big. Never enough time for all the sessions you want to attend or to talk to the people you want to meet. We'll report next month on the jewels we uncover and you can read about the goings on at www.sys-con.com/java.

Java 1.4

Last month Sun officially launched its latest JDK. It will be awhile before it enters mainstream usage, as it takes a significant amount of time before vendors and developers are confident enough to release products that are 1.4+ only. I have an interview scheduled with the senior project manager of 1.4, Sherman Dickman. I've been garnering questions from all around; you may have spotted me asking for them in various mailing lists. Now that I'm armed, I'll be posing these questions to Sherman and you can read the answers in next month's ***JDJ***. One thing that I will comment on with respect to the new 1.4: a lot of people don't seem too bothered about the extra features or API enhancements. A lot of the questions are centered on speed and memory usage. If you've downloaded the new 1.4 and have run some benchmark tests on it, I'd love to hear your views.

How Many Lines of Code?

I've been lurking around some of the Java mailing lists and am continually amused at some of the postings. In fact, let me scrub that, I'm scared to death of some of the posts. There are people out there who obviously believe they're Java programmers and, more important, have probably managed to convince others of this too; it's spiraling out of control, I tell you. We may laugh, but it's these sort of people who have a major impact on our industry and the language. Java doesn't need any more bad press since .NET is on the horizon, and if these people can't get their projects working in Java, their managers will look to alternatives, citing Java as the reason for project failure, not the incompetency of the developer who claimed to have known Java.

Allow me to give you an example that may send you running for the hills. There was an innocent enough question posted on the list regarding JDBC and how to retrieve the number of results in a given ResultSet, preferably without resorting to rewriting your SQL, or performing an additional SQL query such as a COUNT(*). It was a good question and since the API doesn't actually give you a single method for this, it's a fair enough question for someone new to JDBC. The question wasn't the issue; it was the array of answers that tickled me at first until I read the one that scared the life out of me. Before I give you the answer in question, remember the poor guy who posted the problem in the first place. He just wants an answer and is relying on the skillset and expertise of a mailing list to guide him. If I were him, I would be so confused; he'd be better off not asking the question in the first place.

The majority of the answers centered around the [while (Res.next()) noRows++;] answer, which works beautifully. It may not be the most efficient way, but it works. Somebody posted a reply and never twigged it was actually a VisualAge wrapper class

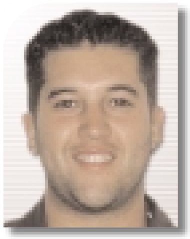
–continued on page 150

alan@sys-con.com

AUTHOR BIO
Alan Williamson is editor-in-chief of Java Developer's Journal. During the day he holds the post of chief technical officer at n-ary (consulting) Ltd, one of the first companies in the UK to specialize in Java at the server side. Rumor has it he welcomes all suggestions and comments.

metrowerks
corp.

www.wireless-studio.com



WRITTEN BY MICHAEL F. SMITH, JR.

Where Do We Go from Here?

Ten years from now we can all reread this editorial reminiscing about "how far we have come." The real question is have we really come that far or do we end up using the same thing, like those old clothes in the closet? I really believe that old shirt of mine from the '80s looks cool.

I've been playing with computers for as long as I can remember, but the real defining moment of my life occurred on May 25, 1995, when I attended the Java Technology Announcement event at Sun. I had been using the Internet, Gopher, FTP, Lynx, Mosaic, and Netscape for some time, but seeing Java applets come to life in a browser enticed me – this was my calling. I went home that night and, with my new-found enthusiasm, started the second Java User Group in the world – Sun beat me to the first one.

Java was born, but at that time its real promise was not readily apparent. It was not going to revolutionize the browser through applets; the real power of Java was how useful it would be for server-side infrastructure. Nevertheless, this was a critical inflection point in our industry.

Fast-forward seven years and we arrive at another critical inflection point in the software industry – at least we like to think so. All we hear about is J2EE, .NET, Web services, and the year of integration. Are we actually in an innovative time that will redefine our industry? Software companies, industry analysts, and the general community of software pundits like to think so. Moving beyond the hype, I still carry a different cell phone in Europe than in the U.S., and my bank still says I can't do everything online.

Granted, the world seems to be focusing on two dominant software infrastructure platforms (J2EE/.NET) and XML is allowing for greater integration between systems. But the software industry is still trying to solve the same old problems. What's changed besides a renewed emphasis on sharing data with more people than ever across the Internet? Does all of this provide us with a fundamental lifestyle change? Hardly. Real innovation, the type that impacts your lifestyle, is yet to come.

The Internet stock boom is over and the venture capitalists are still licking their wounds, taking stock of their wins and losses, and looking for sunnier days. Who or what will bring the momentum back? You, the developer. You hold the keys to unlock the next wave of innovation. What will it be?

I don't have the answer, but when the dust settles on the platform war (J2EE versus .NET), the software industry may have different players, a large number of new applications, and lot more acronyms – but the general problems will likely be the same. After all the dueling analyst reports, the press release wars, the marketing hype, and the battle for your hearts and minds, J2EE and .NET will still be standing, and will likely coexist in most enterprises. The real power of the Internet, XML, and standards is to allow applications to easily talk with other applications. What will you build with this new-found power?

Java has come a long way. Even though J2EE is mightily handling the needs of the enterprise, it must become more accessible, easier to use, and evolve as a total platform. .NET is on the market and has its eyes on the enterprise. But is it ready? Do developers truly want Windows everywhere?

Products will be built, APIs will emerge, standards will be flexed, and software companies will focus on what could be the biggest land grab of the century. Once again, though: What will we do with these platforms and how will we move past the same problems we've faced for 20 years? How do we usher in a new wave of innovation that is truly meaningful?

I don't presume to know what those innovations will be or how they will be built, but I will say it again – developers hold the keys. Developers will be the catalysts that move us beyond solving age-old problems and take us to the next evolutionary level. Will the emergence of J2EE/.NET, XML, and ubiquitous connectivity allow you, the developer, to create lifestyle-changing technologies? I certainly believe so.

Ten years from now we can all look back upon our past innovations. Enjoy the ride.

smitty@bea.com

AUTHOR BIO

Michael F. Smith, Jr., principal technologist at BEA Systems, Inc., is responsible for promoting BEA's Web services strategy and application server technologies. He has been working with Java and Web-related technologies since their early days.

AJIT SAGAR J2EE EDITOR

J2EE Today

By the time you get this issue, JavaOne will be around the corner. Or you picked up this issue at the conference itself. This is JavaOne's seventh year – and for J2EE, it seems that the middle-tier component wars are over, with J2EE clearly emerging as the winning platform for the enterprise.

J2EE has come a long way since the basis of Java's distributed component model, Enterprise JavaBeans, was first established in 1997. The emergence of J2EE as a separate edition of the Java platform has definitely helped to promote it in enterprise applications. However, standard third-party components that can be applied across a variety of applications are still not commonly available.

It's not that easy for someone to pick up a shopping cart component that will integrate with a variety of commerce applications. The J2EE component market is still limited to trees and charts and graphs. Application server vendors offer commerce components, but these are not usually coupled tightly with the app server's environment. For example, leading app server vendors like BEA, IBM, and Sun offer a catalog component. However, despite appearances, I have yet to see one vendor's catalog applied in another vendor's runtime environment. In fact, Sun openly admits that its iPlanet SellerXpert catalog doesn't run on other application servers.

Other vendors like ATG and Macromedia (Allaire) are no longer vying for a place in the J2EE application server race. A couple of years ago, these vendors were touting their own J2EE application servers and offering the complete e-business platform. They still offer J2EE app servers, but these are typically used as example implementations and are not meant for prime time. Now these vendors are content with an integration story that claims compatibility with the leading application servers. In theory, the commerce products offered by these vendors should be able to run on applications developed on J2EE app servers.

The J2EE component market is still developing on several fronts. The good news is that there's enough traction behind J2EE to warrant this growth. App servers are moving toward commodity software; in some cases, they'll end up as a package deal with hardware platforms. Commerce components are emerging as out-of-the-box components that can be used across app servers. We should see more growth in this market; I believe there's enough opportunity in this market for new businesses. In the past couple of years J2EE has established the APIs and components for legacy integration through the Java Connector Architecture, JMS, and Web services components.

This brings me back to JavaOne. J2EE and Web services are the hot topics for the conference with the greatest number of sessions (48 and 63, respectively). The Web EJB 2.0 spec has been out for several months now, and you should see some feedback, good and bad, from folks who have applied it in real-world applications. Enterprise integration through JCA has been gaining momentum over the past year and there are examples of new solution architectures that cover this technology. There will also be several sessions on J2EE 1.3 and the application server market. Web services is closely tied to J2EE in the Java universe. Sessions under this topic will cover Java's XML Pack and the Java XML APIs, as well as building Web services.

With the downturn in the economy, I think we can all expect to see serious real-world application examples, as the industry can no longer afford to experiment on unproven business models. It's gratifying to see that the J2EE platform has withstood the effects of the economy and is forging ahead to pave the way for next-generation enterprise applications.

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 lead architect with Metavonni, LC, based in Dallas, he's well versed in Java, Web, and XML technologies.*

J2EE INDEX

J2EE Today 10
J2EE has come a long way since the basis of Java's distributed component model, Enterprise JavaBeans, was first established in 1997.
by Ajit Sagar

J2EE FAQ 12
The answers to your EJB 2.0 questions

Middleware Becomes a Strategic Commodity 16
Standards, open source, Java, and the Web are combining to force a huge shift in the infrastructure software industry.
by Bob Bickel

Data Mapping to JDBC, XML, and Beyond 18
Abstracting Java's many data source/data sink APIs
by Todd Lauinger

J2EE as the Platform for EAI 26
Supporting open-standard technologies
by Matjaz B. Juric

What Can JCA Adapters Do? 34
Are they the right solution for you?
by David Li

End-to-End Transactionality 42
Myth or reality?
by Mark C. Little

It's a Brave New World Out There 50
The repercussions of the J2EE evolution on the world of application servers
by Ajit Sagar

sun
microsystems

www.sun.com/forte

WHAT IS EJB 2.0?

EJB 2.0 is the latest release of the Enterprise JavaBean specification. The major releases of the EJB specification have been 1.0, 1.1, and 2.0. EJB 2.0 adds several crucial features to version 1.1, including message-driven beans, local interfaces, an enhanced container-managed persistence, and EJB-QL (Query Language).

WHAT ARE THE MAIN INTERFACES IN EJB 2.0?

The figure below illustrates the class hierarchy for EJB 2.0 and the interfaces that are included. The main interfaces for EJBs are packaged under javax.ejb; they're standard extensions to the core Java classes.

The javax.ejb package defines the interfaces you need to extend your application's components. By definition, Enterprise JavaBean is a specification for distributed architecture. In Java terms, this means that a Java class running in one JVM should be able to communicate with an EJB component in another via an RMI call. First, let's look at the EJBObject and EJBHome interfaces, which are shown in the middle pane of the figure. These interfaces extend the java.rmi.Remote interface. The Remote interface serves to identify interfaces whose methods may be invoked from a nonlocal virtual machine. Any object that's a remote object must implement this interface.

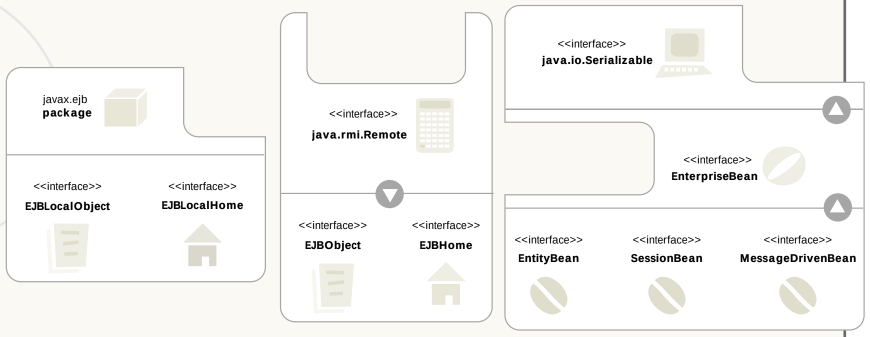
As you probably know, EJBs can only run inside a J2EE EJB container. The container abstracts many of the low-level services from the application developer (you). One set of services that it abstracts is the life-cycle management of the EJBs. The EJBHome interface is used to create this abstraction and can be accessed remotely to create or find the actual EJB component. The methods are invoked by the remote client, but are executed by the container.

The EJBObject interface is used to define the business methods for your EJB component. The interface defines methods to access the actual EJB class and to remove it when it's no longer needed. The EJBObject is a delegator interface that delegates the actual execution of the business objects to the enterprise bean.

The latest release of the EJB specification adds classes and interfaces to support the concept of colocated EJBs (local interfaces). As you can see in the left pane of the figure, the local interfaces (EJBLocalObject and EJBLocalHome) don't extend the java.rmi.Remote interface. You can't use them to access distributed objects. The local interfaces were added to the EJB specification in 2.0 so that component accesses to EJB components within the same JVM don't have to be done through RMI, which is very expensive since each call requires a distributed call.

The enterprise bean interfaces are illustrated on the left side of the figure. These interfaces were added to the EJB specification in 2.0. Local interfaces support the concept of colocated EJBs (local interfaces). The local interfaces (EJBLocalObject and EJBLocalHome) don't extend the java.rmi.Remote interface so you can't use them to access distributed objects. Other than that, the EJBLocalObject is the equivalent of the EJBObject, and the EJBLocalHome is the equivalent of EJBHome. Note that the same EJB component can implement both interfaces simultaneously, thus allowing remote access through RMI and local access through a local method call.

The right pane of the figure shows the interfaces used to create the actual implementation class. The EnterpriseBean interface is a serializable interface for creating the EJB component. The three types of EJBs supported in 2.0 are EntityBean, SessionBean, and MessageDrivenBean. Each of these extends the EntityBean interface.



As an application developer, for each enterprise JavaBean in your application, you need to extend the appropriate home and remote interfaces (local/remote) and provide an implementation for the appropriate enterprise bean (entity/session/messagedrivenbean).

Details on developing the interfaces and classes for your application's EJBs will be covered in subsequent FAQs.

J2EE ROADMAP

The Java 2 Platform, Enterprise Edition defines the APIs for building enterprise-level applications.

J2SE	v. 1.2
Enterprise JavaBeans API	v. 1.1
Java Servlets	v. 2.2
JavaServer Pages Technology	v. 1.1
JDBC Standard Extension	v. 2.0
Java Naming and Directory Interface API	v. 1.2
RMI/IIOP	v. 1.0
Java Transaction API	v. 1.0
JavaMail API	v. 1.1
Java Messaging Service	v. 1.0

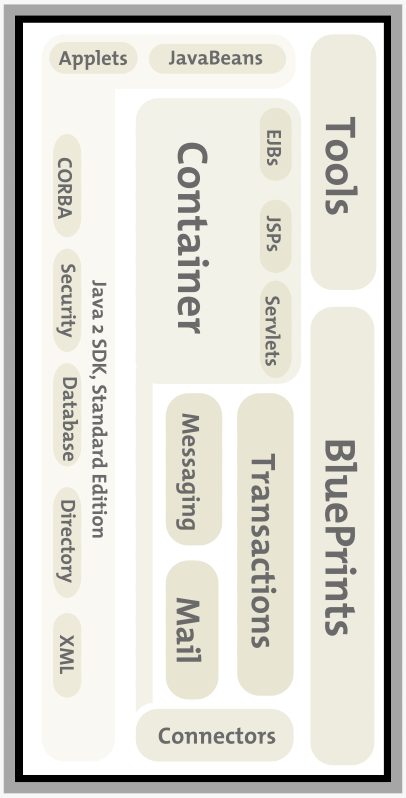
Useful URLs:

Java 2 Platform, Enterprise Edition
www.java.sun.com/j2ee/

J2EE Blueprints
www.java.sun.com/j2ee/blueprints

J2EE Technology Center
http://developer.java.sun.com/developer/products/j2ee/

J2EE Tutorial
http://java.sun.com/j2ee/tutorial/



excelon corporation

www.exln.com

infragistics, inc.

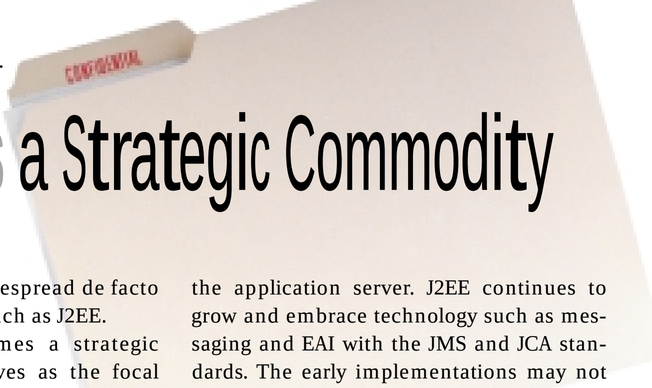
www.infragistics.com

infragistics, inc.

www.infragistics.com

WRITTEN BY BOB BICKEL

Middleware Becomes a Strategic Commodity



Standards, open source, Java, and the Web are combining to force a huge shift in the infrastructure software industry. Middleware is becoming a *strategic commodity*. Free J2EE application servers are just the beginning of this movement. It's crucial to understand that the major forces driving these changes will lead to lower costs, increased collaboration, and reduced time to develop and deploy new applications.

Standards have reached a critical mass where users expect and demand standards-based implementations. Historically, users have preferred to leverage an existing standard to solve a particular problem before expending major funds to develop and deploy new systems. In addition, users will shun vendors for going beyond standards, or not completely implementing them in a timely manner. This forces more industry effort to create and implement these standards – bringing competitors together to bridge differences and arrive at new solutions. Proprietary systems are no longer accepted. This focus on standards minimizes the difference between vendor implementations and has the effect of commoditizing software. However, the effort and investment put forth by vendors to create these standards within the middleware space highlights its strategic importance.

Open source supplements this trend since standards-based APIs are sufficiently defined so that any group of developers can create an implementation. Many developers are being drawn to open source as a direct way of getting their creations to market at a low cost as well as a means to improving the quality of the software by leveraging the resources of the development community. Companies like HP, IBM, and Sun are increasingly dedicating significant engineering resources to many of the open source communities.

While Java is not open sourced, Sun's widespread distribution of the technology and the formation of the Java Community Process has been a loss leader to drive the systems business at Sun. In fact, the Java Community Process is a unique and clever balance that results in widespread de facto standards being created,such as J2EE.

As middleware becomes a strategic commodity, the Web serves as the focal point by:

- Being developed on open source and standards models – HTTP, HTML, and Mosaic
- Enabling easy collaboration for open source development
- Leading to the creation of powerful open source and standards organizations (Apache and W3C)
- Demonstrating that the price for software (e.g., browsers and Web servers) built around open standards would decline sharply and ultimately become free

J2EE Application Servers

Open standards and the resulting open source initiatives have played a major role in the evolution of the J2EE landscape. It's becoming a well-accepted fact that the application server is the new keystone to any significant middleware infrastructure. The Java community's efforts to standardize J2EE have been significant and have yielded a robust, resilient, and enterprise-capable platform, which has resulted in more users adopting the technology while decreasing the difference between application server implementations.

The combination of smaller differentiation and a larger user base is driving down the cost of J2EE application servers. Initially, application server vendors were able to charge huge sums for this strategic software, several times the amount of the accompanying hardware. Open source efforts like Apache Tomcat, JBoss, Orion, and the early Lutris work have proven that application servers can be created through open source initiatives. The resultant software is an attractive alternative for many end users due to its low/free pricing.

Beyond J2EE

It's clear that the concept of middleware becoming a strategic commodity will only accelerate in the future and go well beyond the application server. J2EE continues to grow and embrace technology such as messaging and EAI with the JMS and JCA standards. The early implementations may not stack up yet to the sophistication of proprietary EAI and message bus products, but the low cost of the implementations makes them attractive for many applications. In addition, the amount of investment in these standards will continue to drive the functionality beyond today's proprietary systems with the added benefit of no vendor lock-in, lower training costs, and increased interoperability.

Web services are obviously moving in the same direction. Many of the base pieces – like SOAP, WSDL, and UDDI - are becoming standardized. Users are waiting for higher levels of standardization in such areas as transactions, mediation, and conversational flow control before adopting Web services in a significant way.

The Java community must embrace this trend toward a low-cost strategic commodity to be a viable platform in comparison to Microsoft. The low cost and relative standardization across the Microsoft platform are powerful attraction points. The Java community can't afford to live only at the high end of the market, charging tens of thousands of dollars per CPU of functionality.

The current trend of creating strategic software in the Java community is encouraging – J2EE continues to be the choice for large Web applications. And the new adoption of lower-cost models will help drive widespread adoption of the technology.

Users as diverse as IT departments in small companies to major enterprises and independent software vendors are starting to select J2EE application servers as their strategic deployment choice. The standardization efforts of the JCP and the open source community as well as the low cost are turning the technology into a commodity. This evolution toward a reasonably priced middleware solution significantly augments the transition of users to these new operating environments. The commoditization of Java middleware marks the next pivotal step toward a ubiquitous enterprise deployment platform.

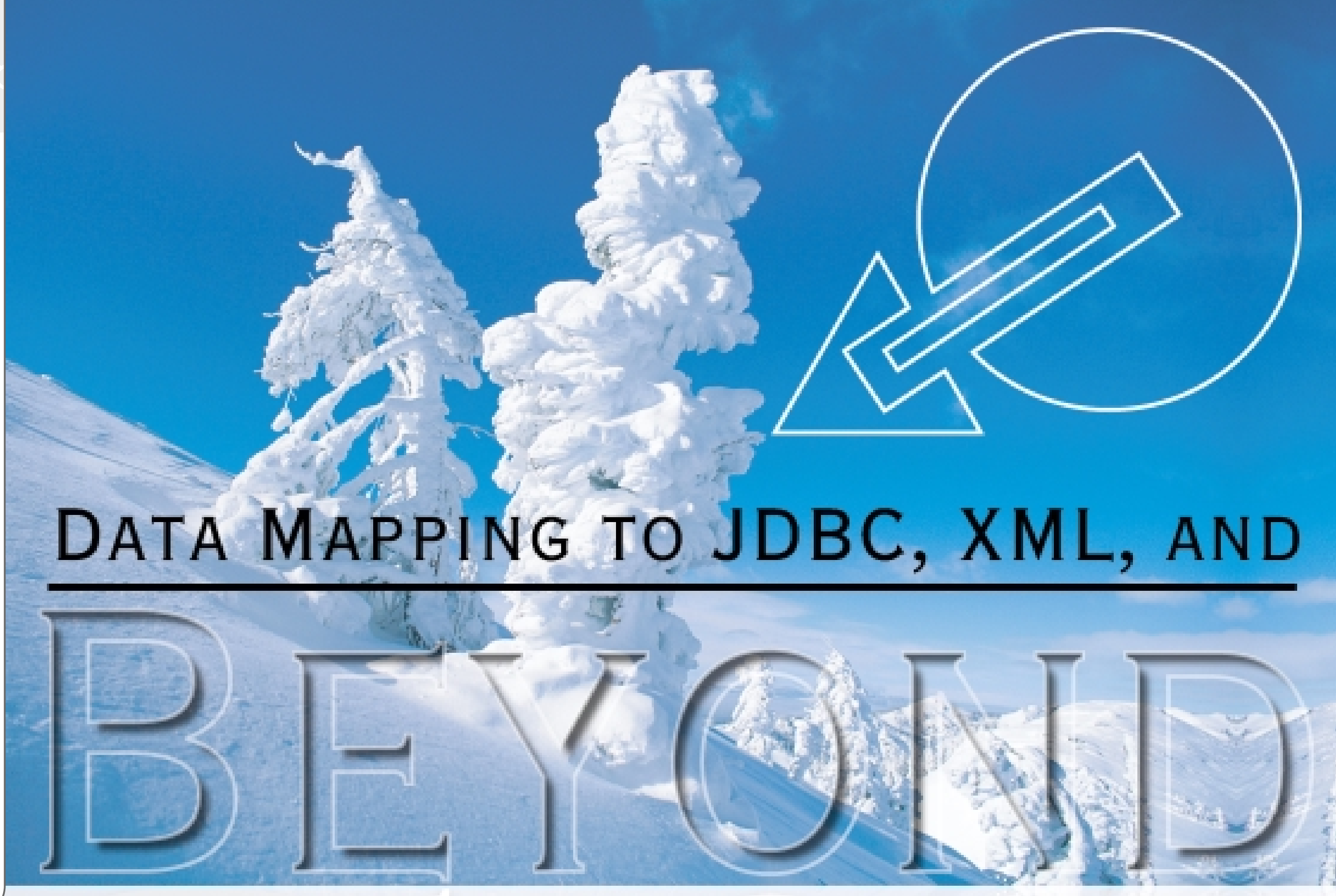
bob_bickel@hp.com

AUTHOR BIO

Bob Bickel, general manager of the Middleware Division at Hewlett-Packard, is one of the original creators of the application server technology. Bob holds a BA in electrical engineering from Bucknell University and an MBA in finance from Temple University.

hewlett-
packard
company

www.hpmiddleware.com/download



Data Mapping to JDBC, XML, and Beyond

WRITTEN BY TODD LAUINGER

SINCE I STARTED MY CAREER IN SOFTWARE DEVELOPMENT, I'VE WITNESSED MANY IMPROVEMENTS IN HOW I DO MY WORK. I'VE GONE FROM ASSEMBLY TO PROCEDURAL LANGUAGES, AND NOW ON TO OBJECT-ORIENTED LANGUAGES.

These paradigm changes have greatly increased my power to express program logic, such that my programs have gotten smaller, simpler, and much easier to understand, while supporting ever-increasing user capabilities. When I started programming, I worked with simple command-line interfaces and text-based "green screens." Next I produced "fat-client" graphical user interfaces, and now I work on Web-enabled user interfaces. Again, each paradigm switch has greatly increased user power, flexibility, and ease of use while the code required to produce the interfaces has decreased and is much simpler to understand.

Data Storage and Retrieval Problems

Unfortunately, I haven't seen the same kinds of advances in data retrieval and storage. In fact, I think we've declined in that area as an increasing number of data source/data sink technologies such as XML, guaranteed messaging, and directory services have come into mainstream development. Besides the user interface, of course, relational data-

This article has been adapted from Chapter 16 of *Beginning Java Databases* by Todd Lauinger. Published by permission of Wrox Press Ltd. and the author.

sybase

www.sybase.com/easerver4dev.

bases used to be the sole data source/sink technology I dealt with. As such, programming environments of the recent past provided first class support. My PowerBuilder and Oracle Forms developer friends have extolled the virtues of these environments over the somewhat primitive JDBC support in Java. My only defense has been the promise of reusable logic in my Java objects that transcend the hard-coded data mappings between PowerBuilder or Oracle Forms screens and the database. Unfortunately, it takes a great deal of JDBC code to map the data involved in the Java objects to the database. Add XML documents, queued messages, and LDAP directories to the mix, and things get even worse. Each of these technologies requires a different Java API, a new learning curve, and a great deal of code to implement. In a recent code survey at my workplace, I found that over 50% of a major application was devoted to nothing but data retrieval, translation, and storage. That left under 50% of the system to do the real work, namely providing a user interface and logic to do something useful with the data users provide us.

Another problem I encountered as I tried to modify and extend the systems at work was the hard-coded data mappings that proliferated throughout. I couldn't add inheritance hierarchies or new classes and relationships. Related classes required inefficient secondary database queries as I moved from one class to another. It was practically impossible to get the existing data mapping code to recognize the need to instantiate the correct subclasses of an object in a class hierarchy as instances were read from a data source.

My most discouraging finding of all was the large number of critical defects in the data storage, manipulation, and retrieval code. There was little care in the placement of transaction boundaries, allowing for all kinds of data integrity problems under less than ideal system operating circumstances. Resources like database connections, statements, and result sets were not being freed correctly, resulting in problems as the application ran over an extended period of time. When processing message queues, the code was committing transactions to the database without a synchronization strategy, such as a two-phase commit, to properly remove messages in the same unit of work. XML documents were not being parsed or generated in an "extensible" way, thus eliminating the crucial X in XML.

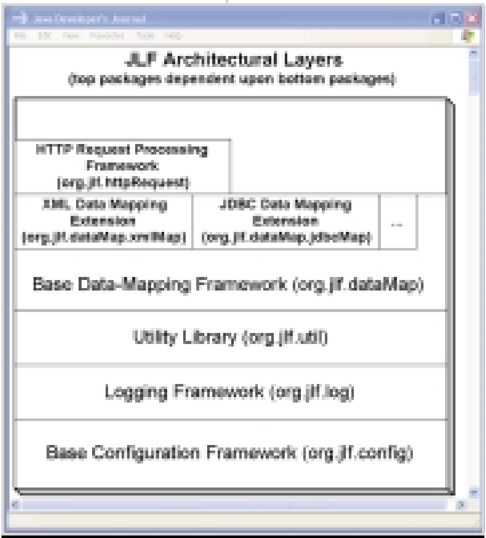

JLF Architectural Layers
(top packages dependent upon bottom packages)
HTTP Request Processing Framework (org.jlf.httpRequest)
XML Data Mapping Extension (org.jlf.dataMap.xmlMap)
JDBC Data Mapping Extension (org.jlf.dataMap.jdbcMap)
...
Base Data-Mapping Framework (org.jlf.dataMap)
Utility Library (org.jlf.util)
Logging Framework (org.jlf.log)
Base Configuration Framework (org.jlf.config)

FIGURE 1 Architectural layer diagram

To solve the problem, I started looking into new Java technologies and APIs like XML data binding, Java data objects, and message-driven EJBs. Each of these technologies had limitations as I tried to hook them up to the logic in my application. Where should I put the logic for objects that crossed data source/data sink technologies? For example, information for my Customer class came in from both the user interface and a message queue, was created or updated in the database, and output as XML documents to the user interface or other enterprise systems. Pretty much every data mapping technology I tried, including the more traditional commercial object/relational mapping frameworks on the market, had either a heavy or exclusive bias to a particular data source/sink technology. I was forced to create multiple Customer classes, one per data source/sink technology (for example, DatabaseCustomer, XMLCustomer, MessageCustomer). Then I'd either have to duplicate the application logic concerned with processing a customer or I'd need to have one Customer class with the logic and transformations to and from the other Customer classes. None of these designs are object-oriented. In responsibility-driven design, a Customer class shouldn't have any logic in it to communicate with a data storage or retrieval mechanism. Instead it should perform the responsibilities of a Customer as abstracted from the problem domain. Other classes in the system should be responsible for the data mapping.

JLF Prototype Data Mapping Framework

Being somewhat of a framework buff, I wondered if I could design a framework that abstracted the dirty details of data source/sink technologies, but provided much of the power and flexibility of the native JDBC, XML, JMS, and JNDI APIs. I came up with the data mapping portion of an open source framework called Java Layered Frameworks (JLF), located at http://jlf.sourceforge.net. This framework works to minimize the amount of code in your application needed to map your Java objects to any number of different data sources/sinks. It also helps you execute complex mappings in a relatively efficient way. For example, when using a JDBC data source/sink, JLF can help reduce the number of SQL statements sent to the database, and it can cache relatively static data so you don't have to read the same data every time you use it.

JLF Data Mapping Overview

JLF is a set of layered frameworks designed to help Java application developers create their applications quicker and with less code. These frameworks include the following capabilities:

1. Configuration framework
2. Logging framework
3. Utility library
4. Data mapping framework
5. HTTP request processing framework

The *configuration framework* basically initializes JLF by identifying where property files are located. Java property files configure the operation of the remainder of the frameworks in JLF, and the configuration framework helps the other frameworks to find those property files.

The *logging framework* is an evolution of my JLog logging framework. It helps to instrument events and log errors in your application so you can detect and correct defects more quickly.

The *utility library* portion of JLF contains code that performs some common coding tasks in Java. Examples include

precise software

www.precise.com/jdj

properly creating hash values for complex objects and using the Reflection API.

The *data mapping framework* is the main framework in JLF and the focus of this article. It's designed to help you map data in your Java objects to any number of different data source/sink technologies. Most of the capabilities of the current version of the framework deal with the JDBC API, but JLF accommodates other types of data sources and sinks as well (for example, output to XML documents or input from servlets). It's also extensible to fit any number of other transactional or nontransactional data source/sink technologies.

The framework layers described above are shown graphically in Figure 1. Each layer shows where the Java package is implemented in parentheses, so you know which package to import in your code.

To use the data mapping framework in JLF, you must understand three core concepts:

1. ***Data mapped objects:*** These are the Java classes you create for your application. They hold the data you want to map to your data source/sink.
2. ***Data mappers:*** The JLF framework provides these objects for you to map your data to and from the data source/sink.
3. ***Data location property files:*** These are the Java property files you create. They tell the data mappers how to map data between the data mapped objects and the data source/sink.

All three concepts go hand-in-hand to accomplish data mapping. We'll now go through each concept in further detail.

Author Bio
Todd Lauinger is a freelance author, conference speaker, teacher, mentor, and consultant. He is currently employed as a software construction fellow at Best Buy Co., Inc., in Minneapolis, MN.

Data Mapped Objects

Any Java classes that you want JLF to map to a data source/sink must be subclasses of JLF's DataMappedObject class. This class contains all the core code to help you define and access variables, relationships, and inheritance hierarchies, so the framework can map these for you. Instead of defining instance variables in your object, define DataAttributeDescriptors. When you want to create relationships between DataMappedObjects in your design, create RelationshipDescriptors. If you have an inheritance hierarchy in your DataMappedObject subclasses, create a hierarchy table so JLF can instantiate the proper types of objects automatically. Figure 2 shows the primary classes in the JLF framework you use to define your DataMappedObjects.

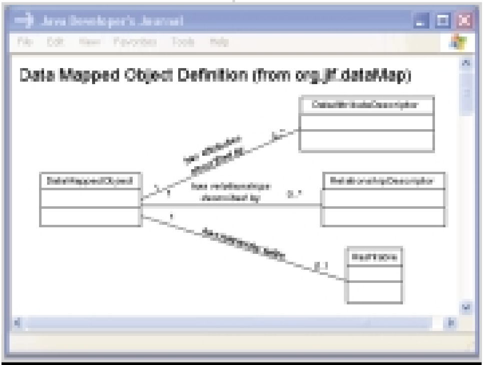


FIGURE 2 Data mapped object definition

Once you've defined your DataMappedObject subclasses with the proper attributes, relationships, and an optional hierarchy table, the data mapped object framework goes to work. It creates DataAttributes and relationships as it maps data back and forth between your Java objects and the database. These two classes of objects help the data mapping framework coordinate the data flowing to and from the database.

DataAttributes are used to replace instance variables in your classes. You may wonder why you can't simply use instance variables like any other JavaBean class would. The answer is twofold. DataAttributes help the data mapping framework efficiently map the data to a database, and they also help to do optimistic locking. In the first case, if you don't change a value in your object after it's read from the database, there's no need to send an update SQL statement when you store your object back to the database. Since you've made no change to the object, sending a SQL statement to the database uses up database resources to change a row to the same values it already contains. Not only would this consume precious database resources, it would also delay application response time to the application user.

The data mapping framework, in the execution of an update() method, first checks to see if anything has really changed in the object before it executes the SQL update statement. If you use simple instance variables in your design, the JLF data mapping framework would have a much more difficult time discovering if you've updated your object. Second, the most efficient way to use a database in a very high-volume transactional system is almost always to use optimistic locking. To use this, execute a locking query before you update or delete an object in the database. The locking query makes sure another process hasn't modified the object since you originally read it from the database. One common way to do this locking query is to check the values of the object in the database and make sure they haven't changed since the original query. With a simple instance variable in your objects, there's no initial value to do the locking query before you update the row with the new value. DataAttributes keep the original value read from the database, as well as the new value that you wish to change the object to.

DataAttributes have different subclasses to help overcome the limitations of Java native types. For example, Java string variables do not have a limit on the number of characters you can store in them. When using a relational database, you almost always define a maximum string length for any of the character columns in your database. The StringAttribute subclass of DataAttribute allows you to define and enforce a maximum string length. Use LongAttribute for int and long variables, DoubleAttribute for float and double variables, DateAttribute for Dates, and, of course, StringAttribute for strings.

Relationship objects help you efficiently map related DataMappedObjects to a database. They help to introduce different database mapping optimizations. For example, you can use them when you deem it more efficient to use one query to populate any number of related Java objects. On the other hand, in cases where you rarely traverse a relationship, you don't want to take the time to populate the objects on the other side of the relationship until you know you need them. Otherwise you'd be inefficiently pulling back large quantities of unused data from the database. The data mapping framework uses relationship objects to "lazy read," or read on demand, such objects when you deem that approach to be more efficient.

Figure 3 shows how the DataAttribute and Relationship objects described earlier work with DataMappedObjects.

datadirect technologies

www.datadirect-technologies.com

Data Mapping to JDBC, XML, and Beyond

Beyond

Data Mappers

The data mapping framework uses a data mapping "plug-in" called a DataMapper. DataMappers map objects to and from a particular data source/sink technology. The goal behind the data mapping plug-in design is to hide the complexity of mapping data to and from that technology. For example, say your Java application needs to map data in its objects to a relational database using the JDBC API, to XML documents using an XML-parsing API, from HTML input forms via the Servlet API, and then send messages to queues using the JMS API. You'd have to learn the complexities of four different and complex APIs to get your work done. You'd also need to write a lot of code, as each API is different and requires completely different code to execute the mapping.

The data mapping framework hides this complexity from you. The code to map your objects to a relational database looks almost identical to the code that maps your objects to an XML document or from the input parameters of a servlet. The DataMapper plug-in deals with the appropriate Java API, so under ideal circumstances your code has no technology-specific API code in it. There will always be cases where the framework doesn't do what you need it to do when using, for example, the JDBCDataMapper. In those cases you write a little bit of JDBC code and hopefully the JDBCDataMapper will do the rest of the work for you. Data mappers in the JLF framework, including the JDBCDataMapper, are shown in Figure 4.

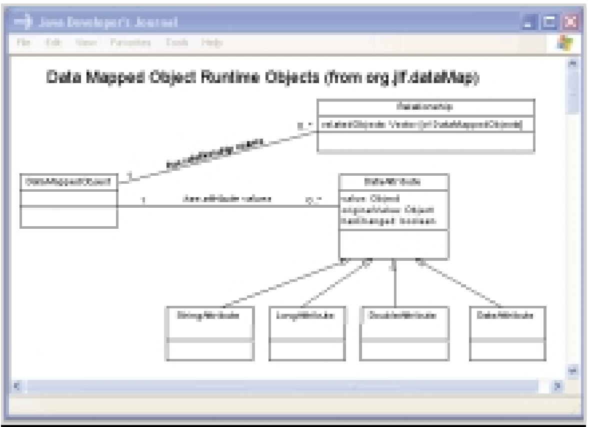


FIGURE 3 Data mapped object runtime object

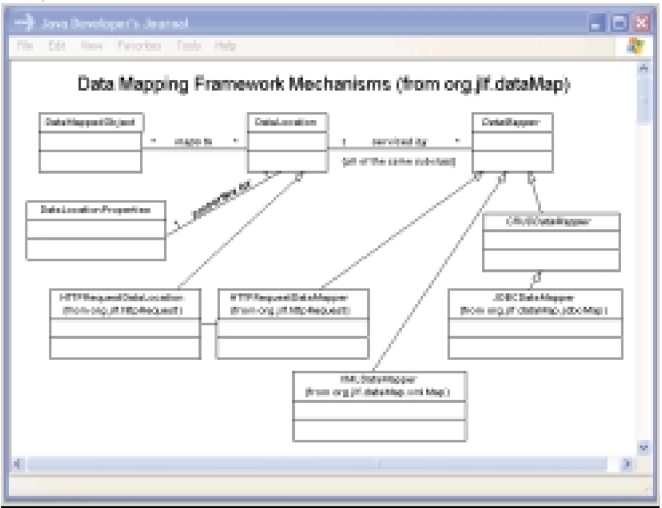

Java Developer's Journal
Data Mapping Framework Mechanisms (from org.jlf.dataMap)
DataMappedObject
maps to
DataLocation
serviced by
[all of the same subclass]
DataMapper
DataLocationProperties
CRUDDataMapper
HTTPRequestDataLocation (from org.jlf.httpRequest)
HTTPRequestDataMapper (from org.jlf.httpRequest)
JDBCDataMapper (from org.jlf.dataMap.jdbcMap)
XMLDataMapper (from org.jlf.dataMap.xmlMap)

FIGURE 4 Data mapping framework mechanisms

Data Location Property Files

Each DataMapper looks to a property file for information on how to map objects to the data source/sink technology it supports. These property files are called *data locations*. They describe how to get to a particular data location and map the data between Java objects and that location. To open a connection to a JDBC data location, the data mapper needs information such as the database URL, the appropriate JDBC driver, and perhaps a user ID and password. Once the connection is established, the data mapper needs to know which SQL statements to send to CRUD (create, read, update, delete) the data. You also tell the data mapper how you want to efficiently map your relationships – reading them in the same query as the original object, or perhaps lazy reading them on demand. In a future article, I hope to explain how each of the data mappers works to rid you of the burdens of data mapping API code.

Conclusion

Enterprise Java software developers, undergoing due diligence in their object design, have a difficult task at hand. Java's APIs for dealing with data sources and data sinks are quite different from technology to technology. The JDBC, XML Parsing, JNDI, and JMS APIs have only the Java programming language in common. As a result, object designers typically hard-code the data mapping between their Java classes and the data source/sink technology they currently deal with. In most cases, this hard-coding is tedious, error-prone, and takes quite a bit of code to carry out.

Inheritance hierarchies, involved in almost any nontrivial object design, are typically abandoned because of data mapping difficulties. In addition, if the data source/sink design changes, it has a direct impact on the Java code (for example, the Java code is tightly coupled to the design of a database). When the same Java class needs to communicate with another data source/sink technology, it's often easier to start from scratch rather than incorporate a second data source/sink mapping into the current class.

The JLF data mapping framework tries to address all these problems by separating the design of your Java classes from the mapping of them to and from a data source/sink. JLF abstracts the details of executing different technology mappings using data mappers. It provides default implementations of JDBC, XML (currently write only), and servlet data mappers and is hopefully extensible for you to add your own. This should leave you free to concentrate on good object design instead of dealing with all of Java's data mapping APIs.

todd.lauinger@bigfoot.com

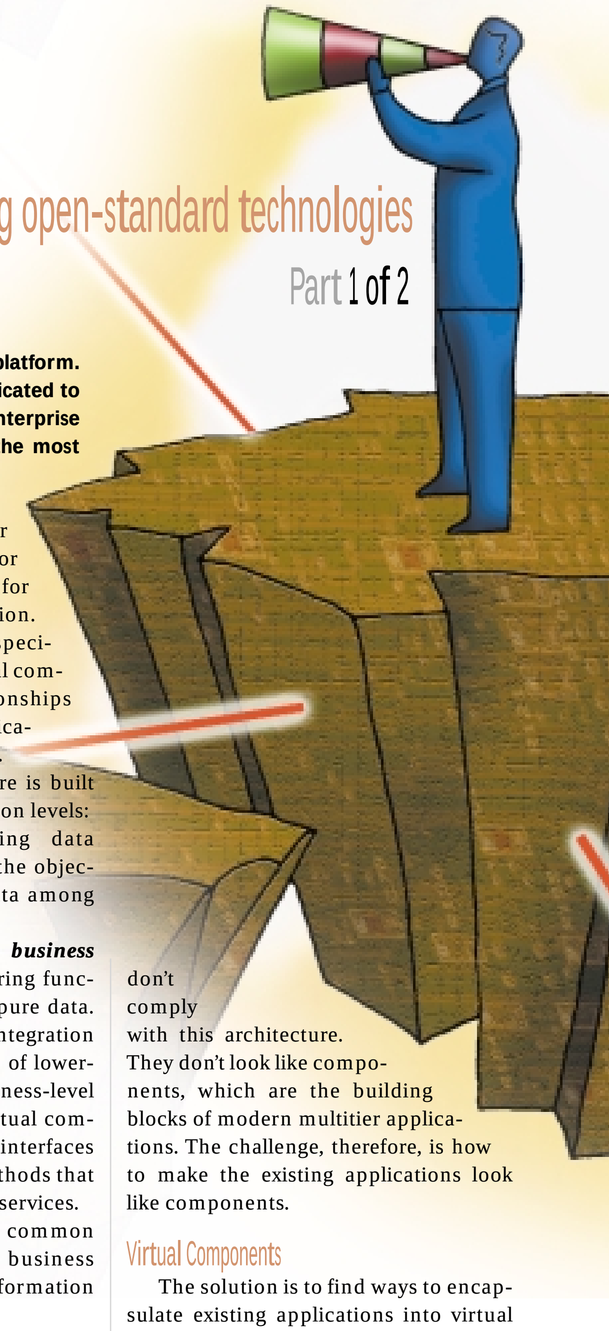
sitraka
software

www.sitraka.com/jclass/jdj

J2EE as the Platform for EAI

Supporting open-standard technologies

Part 1 of 2



WRITTEN BY
Matjaz B. Juric

Java was not originally designed as an enterprise platform. However, over the years it gained a new functionality dedicated to enterprise computing that became the Java 2 Platform, Enterprise Edition (J2EE). J2EE, currently in version 1.3, is one of the most important modern enterprise platforms.

An enterprise platform has to provide ways to integrate with existing systems and applications. The fact is, most companies have applications and they don't exist in isolation. New applications developed on the J2EE platform need to be integrated with other applications. Although this might sound relatively simple, we must be aware that companies need fully integrated information systems. This means that all applications, no matter which technology they're developed in, function as a large integrated system. The total integration of applications within a company is referred to as Enterprise Application Integration (EAI).

Although we won't go into details, for successful EAI we have to select the integration infrastructure technologies and define the integration architecture.

Effective integration infrastructure should address relevant middleware technologies that are required for EAI. Frequently we'll have to use more than a single middleware technology. When selecting and mixing different technologies we have to focus on their interoperability. Achieving interoperability between technologies can be difficult even for those based on open standards; for proprietary solutions interoperability is even more difficult.

It's not only a question of whether we can achieve interoperability but of how much effort we have to put in to achieve it. Therefore, companies are leaning more toward software platforms, such as J2EE, that gather compatible technologies. The J2EE platform provides a number of protocols and technology specifications for these purposes. The most important protocols are HTTP(S) and IIOP. The most important middleware APIs are JDBC, RMI-IIOP, Java IDL, JMS, Java Connector Architecture, and JNDI. Very important APIs, which we'll discuss in detail later, are also JAXP for XML support, JTA and JTS for transaction support, and JAAS for authentication and authorization.

Integration architecture specifies the overall structure, logical components, and logical relationships between all the different applications we want to integrate. Usually integration architecture is built through the following integration levels:

- ***Data:*** Focuses on moving data between applications with the objective of sharing the same data among the different applications.
- ***Application interface and business method:*** Both focus on sharing functionality and data, not just pure data. Application interface-level integration is achieved through the use of lower-level APIs. The goal of business-level integration is to develop virtual components that will provide interfaces with high-level business methods that can be considered business services.
- ***Presentation:*** Implements a common user interface for the business method–level integrated information system.

The integration architecture should provide the basis for an enterprise-wide information system that will benefit from the synergy provided by the integration of different applications. We have to define an architecture that makes it possible to reuse existing applications. And the architecture should easily accommodate new generation applications, providing a scalable architecture that will allow us to easily make changes and modifications in the future.

There's nothing controversial in stating that we should use the multitier architecture. The problem is that most existing applications (particularly legacy) don't comply with this architecture. They don't look like components, which are the building blocks of modern multitier applications. The challenge, therefore, is how to make the existing applications look like components.

Virtual Components

The solution is to find ways to encapsulate existing applications into virtual reusable components that will provide several interfaces and expose the functionality in a universal manner. To define and implement virtual components, we'll follow the runtime reusability of the business logic implemented in the existing applications and delegate operations to one or more existing applications.

When accessing these virtual components through the interfaces it doesn't matter whether we're dealing with a newly developed component or a virtual component that encapsulates an existing application; we'll access both in the same manner. Figure 1 shows the business logic tier with two virtual components that reuse existing applications

canoo
engineering ag

www.canoo.com/ulc/

as well as a newly developed component, all exposing their functionality through interfaces.

On one side, the virtual component presents the existing application through abstract interoperability interfaces; on the other, it communicates with the existing application using the existing facilities. Virtual components with abstract interoperability interfaces present an application view that's implementation-independent. If we keep the abstract interface constant, the other applications are not aware of any changes made on the original application. Thus, when a change is made, there's no need to modify other applications that depend on the original application. In other words, client virtual components can't determine how the server virtual components are implemented.

An integrated information system based on virtual components looks like a newly developed information system, as it actually reuses the functionality of existing applications. Note that in each integration level we'll build more complex and abstract virtual components:

- ***Data:*** Virtual components will provide access to the data only.
- ***Application interface:*** Virtual components enable you to reuse not only the data, but also the functionality of existing applications. However, the reuse of functionality takes place at a low abstraction level, and basically resembles the application programming interfaces built into existing applications.
- ***Business method:*** Raises the abstraction level of virtual components to the level where the interfaces will provide high-level business functions.
- ***Presentation:*** Adds a newly developed presentation tier on top and makes the integrated composite information system look like a newly developed information system.

Implementing Virtual Components in J2EE

As we know, J2EE is built on the multitier application model and components are the building blocks of each tier.

- The components for the client tier are application clients and applets.
- The components for the Web component tier are servlets and JSP pages that provide the ability to service thin clients. They respond to HTTP requests and generate thin-client user interfaces using HTML, XML, and other technologies.
- The components for the business logic tier are EJBs. These provide an environment that supports services such as transactions, security, and persistence.

The only tier that doesn't consist of components is the EIS tier. This tier contains the existing enterprise infrastructure, such as ERP systems, TP monitors, and database management systems.

J2EE supports several open standards for communication between tiers. This includes HTTP(S) and IIOP protocols and APIs, such as JDBC, JMS, JNDI, and Java Connector Architecture. This enables us to include components that haven't been developed in Java on practically any other tier, thus providing a way to develop virtual components for existing applications.

On the business logic tier we can deploy EJB components as well as CORBA (and RMI-IIOP) distributed

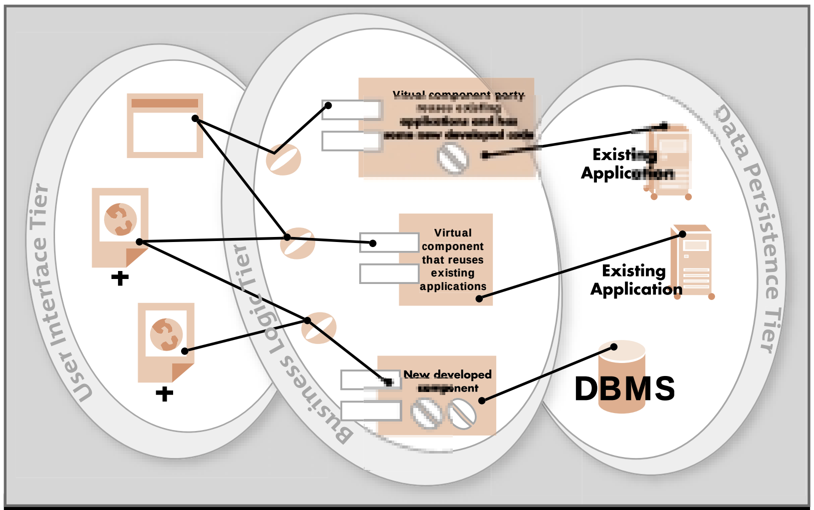

User Interface Tier
Business Logic Tier
Data Persistence Tier
Vitual component party reuses existing applications and has some new developed code
Virtual component that reuses existing applications
New developed component
Existing Application
Existing Application
DBMS

FIGURE 1 Virtual components

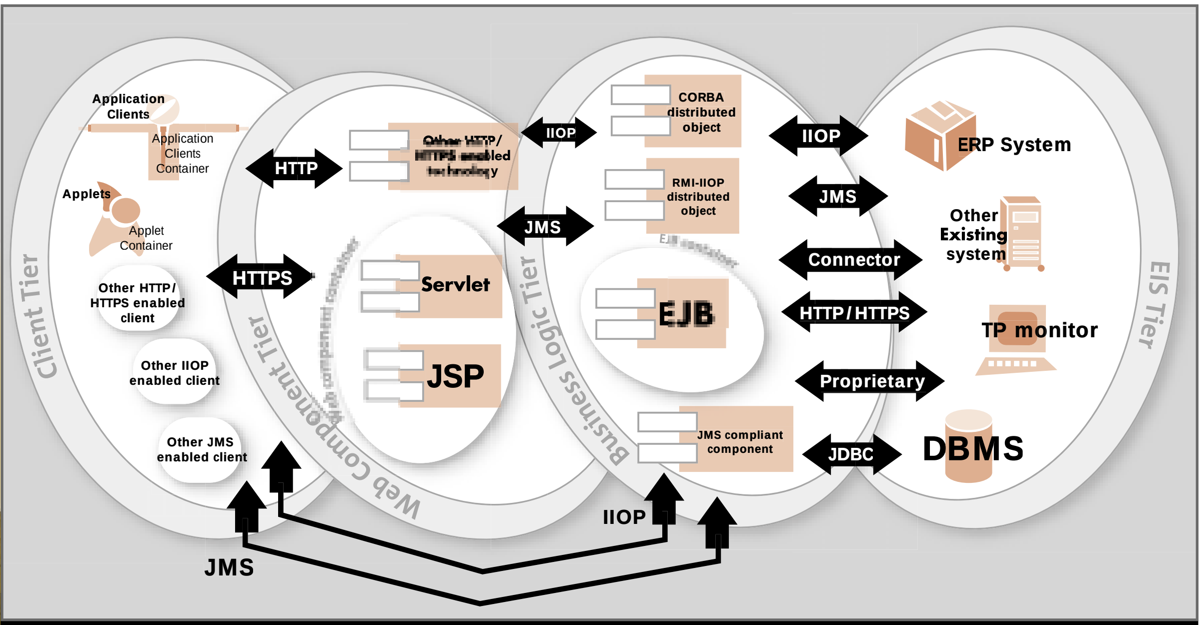


FIGURE 2 J2EE integration architecture

esri

www.esri.com/arcims

ibm

ibm.com / websphere / ibmtools

objects and components that communicate through a JMS-compliant MOM. This is one of the most important facts for integration. We can see that for developing virtual components on the business logic tier we're not limited to EJBs, but can use CORBA, RMI-IIOP, and MOM components too. Please note that CORBA and MOM components don't have to be developed in Java, rather we can use a variety of programming languages. Both CORBA and leading MOM products support all popular programming languages, including C++, C, Smalltalk, Ada, COBOL, Delphi, and even Visual Basic.

Since these non-Java components don't reside in the EJB container, they can't take advantage of the managed environment provided by containers. However, they can still take part in transaction and security mechanisms. For this they have to use the corresponding APIs, but we'll come to this later.

On the Web component tier we can also deploy other non-Java components. These include any components that can respond to HTTP(S) requests. Again, we can't execute them into the Web container provided by the J2EE application server. However, we usually won't be accessing existing applications directly from the Web component tier but via the business logic tier.

The same is true for the client tier – the application clients are not necessarily Java applications. They can be developed in other languages and use the identified protocols to communicate with the other tiers. Examples include Web browsers, CORBA clients, and MOM clients.

This brings us to the extended J2EE integration architecture that we'll use to achieve J2EE EAI, shown conceptually in Figure 2. The extended J2EE integration architecture cleanly separates the tiers and places a limited set of protocols that can be used for communication between the tiers. The business logic tier separates the Web component and the client tier from the EIS tier. The client and Web component tier shouldn't access the EIS tier.

The only way to access the EIS tier is from the business logic tier. Here the EJB, CORBA, RMI-IIOP, and JMS-compliant components use a variety of protocols and technologies to access the EIS tier. Only here we're faced with the heterogeneity of the existing applications located in the EIS tier. The business logic tier thus implements the Façade pattern for the Web component and client tiers. Through this Façade, the business logic tier provides access to business functionality through clearly defined interfaces.

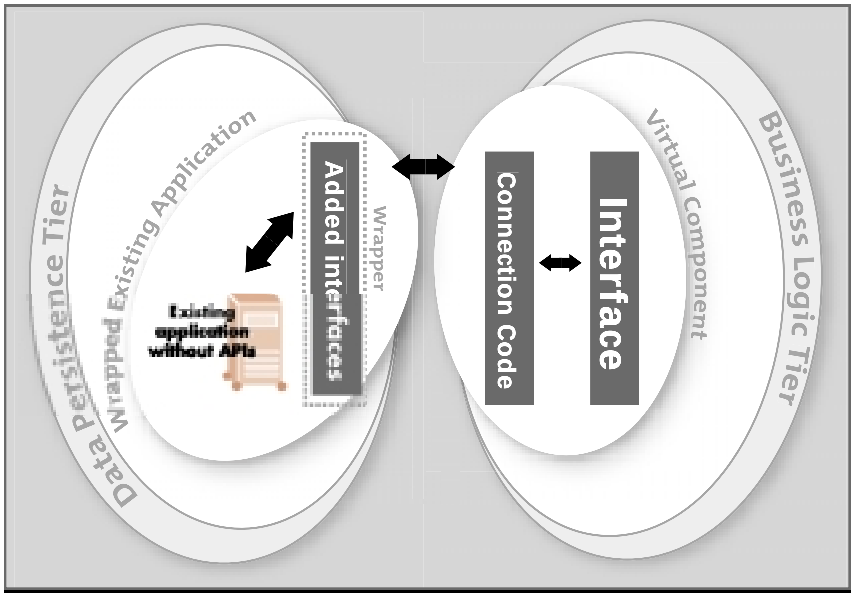


FIGURE 3 Application wrappers and virtual components

Accessing Existing Applications

Virtual components are not very difficult to implement, especially if existing applications already provide some APIs through which we can access the required functionality programmatically. Then we don't even have to modify existing applications.

If the existing applications don't provide any APIs or if they're provided but don't meet our requirements, we'll have to define and build them. For this we'll modify the existing application and add the necessary APIs. We'll call the API a *wrapper* and the modified existing application will be referred to as a *wrapped existing application*.

Adding wrappers can be quite straightforward, especially in solutions developed in-house and where source code and documentation is available. However, if we

ibm
ibm.com/web-sphere/ibmtools

J2ME J2SE J2EE Home

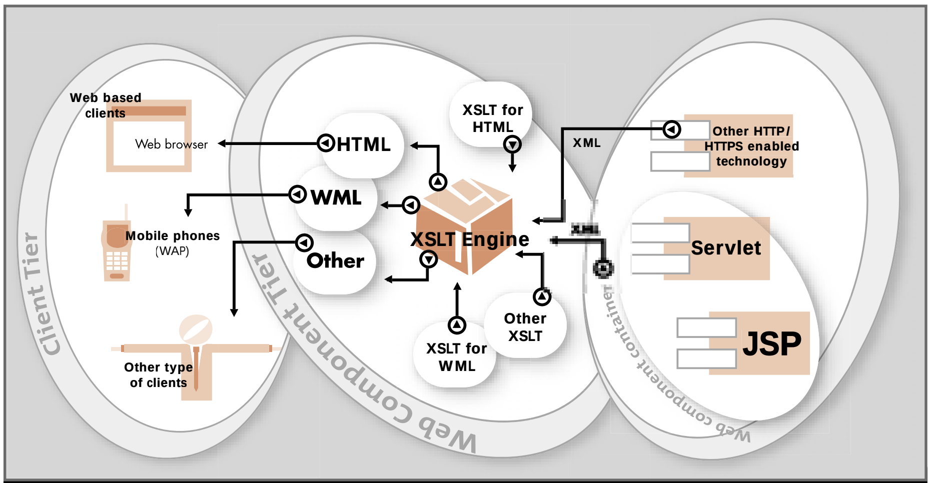
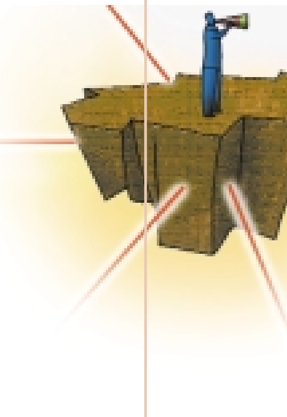


FIGURE 4 Support for different types of clients

don't have the source code and the documentation is limited, changing the application can be difficult.

When faced with such a situation, we can consider using direct database access or even the user interface to access the functionality. Techniques like screen scraping and terminal emulation, where wrappers access the application functionality through the user interface, can help us get the necessary information without modifying the applications. CORBA and JMS are the most frequently used technologies for developing wrappers in J2EE. Wrappers are schematically shown in Figure 3.



Support for XML

Although I've discussed the integration architecture, I didn't mention the formats that can be used for exchanging data between applications. In the last few years XML has established itself as the de facto standard for data exchange.

The power of XML lies in the possibility that we can define the tags ourselves, enabling us to describe any data in any structure (although XML is perhaps best suited to hierarchical data structures). However, freely defined tags can cause semantic problems in the data exchange. It's impossible for a computer to "guess" the meaning of the data from the tags. A vocabulary agreement must be achieved between the parties in which each party has to support the same set of tags in the form of a clearly defined DTD or Schema. Achieving an agreement on vocabulary inside the company is not as difficult as between companies. Therefore great effort is put into the definition of standardized vocabularies for different industries.

J2EE v.1.3 provides support for XML through the Java API for XML Processing (JAXP). JAXP enables you to verify, create, access, and manipulate XML documents from Java components. To manipulate an XML document, it must first be parsed. In the process of parsing, the document is often validated. Then the content of the document must be made available through an API. Currently, two important APIs are supported – the Document Object Model (DOM) and the Simple API for XML (SAX) – by JAXP. JAXP 1.1 also includes an XSLT framework based on the Transformation API for XML (TrAX).

Support for XML, provided through JAXP, is adequate for EAI. Still, Sun's Java Community Program is currently defining additional XML-related specifications, most of them focused on Web services, including:

- ***JAXM (JSR 67) – Java API for XML Messaging:*** Provides an API for packaging and transporting document-oriented XML messages using on-the-wire protocols defined by ebXML.org, Oasis, W3C, and IETF
- ***JAX/RPC (JSR 101) – Java API for XML -Based Remote Procedure Calls:*** Supports XML-based RPC standards
- ***JAXR (JSR 93) – Java API for XML Registries:*** Provides an API for a set of distributed registry services that enables business-to-business integration between business enterprises using the protocols defined by ebXML.org
- ***JWSDL (JSR 110) – Java Web Service Definition Language:*** Provides a standard set of APIs for representing and manipulating services described by Web Services Description Language (WSDL) documents. These APIs define a way to construct and manipulate models of service description
- ***JAXB (JSR 31) – Java API for XML Binding:*** Allows us to compile an XML Schema into one or more Java classes that can parse, generate, and validate documents that follow the Schema

Support for Different Types of Clients

Integration architecture is also concerned with providing support for different types of clients. In fact, the integration architecture already supports different types of clients: rich GUI and Web. Web clients use the Web component tier, which generates the presentation for the client tier. Today the most common approach is to generate HTML pages.

However, this technique can't fulfill the increasing requirements to support other types of clients, such as Palm top computers, mobile phones, or any other device that enables client access. While there's no hindrance in developing different Web components that would generate different presentation formats, this solution is costly because it requires the additional development of Web components for each distinct client type. These Web components also contain very similar logic – they differ only in the way they present the data, which introduces maintenance problems.

The solution is to generate the content in a universal way only once, and then enrich it with the appropriate presentation information in order to present it to the client.

Unfortunately, version 1.3 of the J2EE platform doesn't provide a technology that would make this procedure automatic, as does Microsoft .NET. But in J2EE we can still reach a high-level of automatism with its built-in support for XML. Rather than generating HTML pages on the Web component tier, we can generate XML, which then has to be transformed to a presentation appropriate for the client tier. This can be HTML for Web browsers, WML for WAP devices, or any other appropriate format. The transformations between these formats can be done with Extensible Stylesheet Language for Transformations (XSLT) supported by JAXP. In addition to XML, the XSLT engine requires a stylesheet that defines how different parts of the document will be represented on the client. The XSLT engine then performs the transformation (see Figure 4).

Summary

J2EE is a very suitable platform for EAI as it provides support for open-standard technologies and protocols that enable integration with existing applications. It also provides support for XML and related technologies, which we can use for data exchange and for supporting different types of clients.

Resource

- Juric, M.B., with Basha, S.J., Leander, R., and Nagappan, R. (2001). *Professional J2EE EAI*. Wrox Press.

AUTHOR BIO

Matjaz B. Juric is an assistant professor at the University of Maribor, and is the author of Professional J2EE EAI and coauthor of Professional EJB. Matjaz holds a PhD in computer and information science.

matjaz.juric@uni-mb.si

ibm

ibm.com/db2/outperform

What Can JCA Adapters Do?

Are they the right solution for you?

Sun released the J2EE Connector Architecture (JCA) 1.0 specification in August 2001. The JCA provides a general framework that standardizes the programming interface and the quality of services (QoS) implementation interface for an enterprise information system (EIS). Since then, it's rapidly gaining popularity among EIS and integration tool vendors, and many early releases of the JCA adapters are available for a variety of EIS applications.

Written by David Li

Whether you're an IT manager or a J2EE architect, if you're interested in EIS connectivity you'll be excited about the promises of JCA. What is JCA? What are its most appealing features? What are its shortcomings? Who are the vendors that support it? Are there any other choices so I can do some comparison shopping? What path should I take from here? This article sheds some light on these questions.

First I provide an overview of JCA and its features, then I discuss the type of support provided by the application servers in the market. The rest of this article focuses on the state of the market for JCA adapters. Finally I offer some guidelines to help you determine if JCA is the right solution for you.

JCA Features

Before JCA became available, connectivity to EIS presented a set of common issues and needs. First, each EIS application had its own programming interface. Talking to a different EIS application required programming against a different set of APIs. A common set of client interfaces was needed to simplify the programming effort. Second, the traffic to a back-end EIS application would likely be heavy. You needed connection pooling to cut down connection-related overhead and enhance performance. Third, the connectivity to an EIS application was often transaction-oriented. You needed built-in transaction support to guarantee data integrity with the least amount of programming effort. Last but not least, security integration across EIS applications and EIS clients was highly desirable.

If you think about these issues, they're similar to the ones regarding connectivity to very early databases. These problems are now resolved due to widely adopted technologies like JDBC. As a programmer, you don't need to talk directly to a database but you can through the JDBC API, which is the same across all popular databases. You can use connection pooling to a database but you don't need to implement it. You can use transaction support and security integration easily since support for these features is built-in. Wouldn't it be great to have a similar technology for EIS applications as well? If your answer is yes, JCA is the answer:

JCA resolves these issues by providing the following features:

- ***Connection pooling:*** An EIS connection is usually expensive and time-consuming to create. Connection pooling enables application servers to create and share connections to EIS applications so that expensive connection resources can be used efficiently. JCA adapters and J2EE application servers implement this service.
- ***Transaction management:*** This enables an EIS application to be enlisted in the transaction context provided by an application server so that this server can manage the transaction of the EAI system as one unit. JCA adapters and J2EE application servers also implement this service.
- ***Security:*** Security interface implementation allows application servers to manage overall security without compromising an EIS-specific security mechanism. Authentication, authorization, and secure association are the areas covered by this interface. This is a built-in service for JCA adapters and J2EE application servers.

blaze advisor
from hnc

www.blazesoft.com

- ***Common Client Interface:*** JCA also defines a user-level programming interface referred to as the common client interface (CCI). This interface set is optional in JCA 1.0 and allows developers of an EIS client to connect, interact (execute a command), and get a result set from the targeted EIS in a standard way.

JCA Support of Application Servers

JCA receives support from two places: a JCA supporting application server and a JCA supporting adapter to an EIS application. JCA 1.0 is part of the J2EE 1.3 specification and a J2EE 1.3–compliant application server must provide the environment to support the required JCA features such as pooling, transaction, and integrated security. Table 1 provides a list of commonly used application servers and their JCA support status.

BEA's WebLogic Server is one of the early supporters of JCA. JCA beta support was built into WebLogic 6.0 in the beginning of 2001 when the JCA 1.0 specification was in its final draft. With about one year to mature, the award-winning WebLogic Server is one of the best application servers with JCA support. IBM WebSphere Application Server is another popular and award-winning J2EE app server. Its JCA support became available around mid 2001. JBoss is another one that's worth a special mention. If you're under a tight budget, you may want to look into this product. It supports JCA and you can't beat its price – free!

Vendor Name	J2EE Application Server	Availability
Allaire Corporation	JRun	Not available
Bluestone (HP)	Total-e-Server 7.3 (MP2 and above)	Available
BEA	WebLogic Server 6.1	Available
Borland	Borland Enterprise Server (AppServer Edition)	Available
IBM	WebSphere Application Server 4.0 (Advanced Edition)	Available
iPlanet	iPlanet Application Server	Not Available
JBoss	JBoss 2.4.4	Available
Oracle	Oracle9i Application Server Release 2	Available
Pramati	Pramati Server 3.0	Available

Table 1 JCA support status

Adapter Vendors

You also need a JCA-compliant adapter to connect to your back-end EIS application. Many integration vendors have built JCA-compliant adapters. Table 2 provides a compiled list of the vendors and the JCA adapters.

In this table, many vendors have JCA adapters for the same EIS application. However, a JCA adapter from one vendor may support a different set of features from the others even if it's for the same EIS application. This is caused by two factors. (1) Some specifications such as the CCI in JCA 1.0 is optional; it's up to an adapter vendor to decide whether they want to include this feature in the current release. (2) Some of the important features for EIS integration are not included in the current JCA specification. Adapter vendors may decide to add additional features to enhance an adapter's functionality. The missing features will be discussed in more detail later.

Since these missing features from the JCA specification can be important ones, most of the vendors surveyed here took a more practical approach and stayed ahead of the curve. One or more additional features were added to their adapters to provide extra functionality even though their implementations are proprietary.

Insevo has JCA adapters for many EIS applications including SAP, PeopleSoft, JD Edwards, and Siebal. These adapters support an XML-based interface in addition to the JCA-defined CCI. They support both synchronous and asynchronous communications between a client and EIS applications. They also support bidirectional communication instead of JCA-defined single direction communication. These additional features make Insevo's adapters not only suitable for application integration but also for process integration. These additional features are already being considered as part of the JCA 2.0 specification. So in some sense, Insevo's adapters are one version ahead of the JCA specification. The additional features are not JCA compliant, but if you really need the features can you get anything better?

RAi connectors by Resource Adapters are another set of JCA adapters that took the same approach. They incorporate some of JCA 2.0's anticipated features. RAi supports both inbound and outbound connectivity as well as synchronous and asynchronous communication modes. In addition to CCI, RAi connectors support XML-based APIs and XML metadata and provide a logging and monitoring utility, which can be a big plus in real life.

In addition, Attunity and Insevo also provide numerous data source and legacy adapters. These adapters usually need only a single direction and a synchronous type of communication. Some of the data sources and legacy systems don't support JCA features such as transactions. As a result, not all JCA features are fully supported.

"The features provided by Web services and JCA complement each other. It won't be surprising if these two technologies eventually merge their features"

Comparison with Other Kinds of Adapters

Other types of adapters are developed for different requirements. One new and important type is a Web services adapter that's rapidly gaining popularity as well. Web services is an HTTP/SOAP-based integration interface and an XML-based

altoweb

www.altoweb.com

protocol. And a lot of proprietary adapters were developed prior to the introduction of JCA, so these adapters had a longer time to mature.

Web Services Adapter

Nowadays, an enterprise uses all types of platforms some of which are Java-based. Companies invest a large number of resources to develop their systems and usually are not in a position to abandon them. The question is how to integrate these heterogeneous systems so they all work together without an additional investment. Fortunately two very popular technologies make it possible. One is HTTP protocol and the other is XML. They're the technologies that every platform uses and are a great fit for heterogeneous system integration. Web services is a specification that is built on these two simple yet critical technologies. While the full discussion of Web services is beyond the scope of this article, here's a short list of Web services features.

- ***XML interface:*** Web services is XML based. It uses the Web Services Description Language (WSDL) to describe the service format of the end-point service provider
- ***HTTP/HTTPS protocol:*** The de facto protocol for Web services
- ***SOAP:*** The binding protocol between WSDL-based Web services and HTTP/HTTPS communication protocol

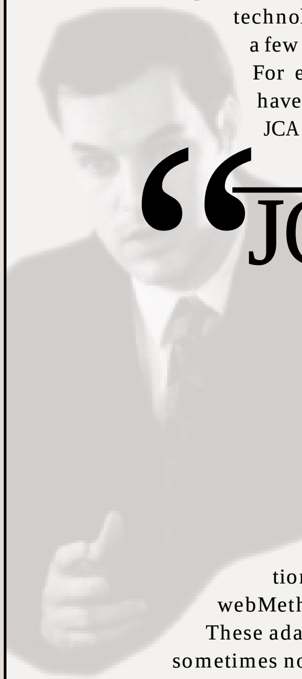
Web services doesn't provide any QoS yet and is a synchronous protocol. It's a good fit for loosely coupled heterogeneous system integration.

The features provided by Web services and JCA complement each other. It won't be surprising if these two technologies eventually merge their features. In fact, a few vendors have already moved in that direction. For example, vendors like Attunity and Sirvisetti have already built Web services support into their JCA adapters.

EIS supported	JCA Vendor Name
Baan	Insevo
SAP	SAP, Actional, Attunity, Insevo, Resource Adapters, Sirvisetti
PeopleSoft	PeopleSoft, Actional, Insevo, Resource Adapters,
IBM CICS	IBM, Insevo, Comporsys Hansa GmbH
Siebel	Siebel, Actional, Insevo, Resource Adapters
Oracle	Insevo, Resource Adapters, Sirvisetti
JD Edwards	JD Edwards, Insevo

Table 2 Vendors and JCA adapters

“JCA has been widely adopted as an open-industry standard and will flourish in the future. It is architecture neutral, and cheaper and easier to implement than most proprietary solutions”



Proprietary Adapter

Prior to the introduction of JCA, integration adapters from independent vendors like webMethods and TIBCO have been available for years. These adapters will likely have proprietary APIs that are sometimes nondetachable from their integration packages. However, these adapters have also been tested for years and have much wider EIS coverage than JCA adapters. In particular, the webMethods Enterprise Adapter and B2B adapters have the most coverage by far. webMethods has more than 60 adapters. These adapters don't support JCA yet, even though webMethods is rapidly moving to support it. Table 3 provides a list of some of these adapters.

Short Term Solution / Long Term Strategy

Pros and Cons

The advantages of JCA are obvious. It provides EIS vendors with a way to expose an EIS interface in an open-industry standard. By using the common callable interface and inheriting the QoS provided by JCA, a programmer can simplify the EIS integration effort without sacrificing performance and system integrity.

The limitations of JCA are less obvious but can be important. Like any new technology the immaturity of the initial implementation is often the most frustrating problem to deal with. One specific issue is that JCA adapters should be portable across application servers. However, at this point, this statement may not be true for your application server. The adapter's support for an application server is tested and released by the adapter vendor on a case-by-case basis.

In addition, there are a few known limitations, some of which will be addressed in the future version of the JCA standard.

- ***Asynchronous message delivery:*** JCA 1.0 tackles a synchronous call to EIS applications. It doesn't handle asynchronous message delivery to and from EIS applications. If asynchronous delivery is a requirement for your company, you may want to use JCA in combination with the Java Message Service (JMS) or another queuing service. The other choice is to use JCA adapters with proprietary asynchronous communication support built in.
- ***Long-running transaction:*** This is the type of transaction that may run for days or even months. It doesn't achieve data integrity by locking resources but by long-term coordination of resources to guarantee the successful execution of a workflow. Long-term transactions are of increasing importance due to the growing need for B2B integration, and that the time span of a process often can't be controlled by a single trading partner. JCA 1.0 supports DTC-type short-term transactions but it doesn't support any long-running transactions. If you're thinking of using JCA to integrate with your trading partner directly, it may not be the best fit. You're better off using it inside an enterprise and inserting a workflow engine between you and your trading partners for long-running transactions.
- ***No XML-based interface:*** There's no XML-based interface to provide interoperability with non Java–based heterogeneous

altova

www.altova.com

Type	Name
Packaged adapter	3270
	Clarify
	SAP R/3 ALE/IDOC
	Sybase
	Vantive
	Oracle
	IBM MQ Series
	Informix
API-based adapter	COM/COM+
	C/C++
	Java
	C for MPE/iX
	CORBA/IDL
Third-party adapter	Acxiom
	Forte
	Pivotal
	Saviion
	Baan ERP
	Selectica
	Calico
	Point Information System
B2B Adapter	Baan Edition
	Development Kit
	Enterprise JavaBeans Edition
	J.D. Edwards' WorldSoftware Edition
	MSMQ Edition
	Oracle Applications Edition
	PeopleSoft Edition
	SAP Software Edition

Table 3 Proprietary adapters

systems. The result set returned from the CCI is row- and column-based. It's similar to the result set from a JDBC call to a database. The data returned from the rows and columns are in their native data format. Additional XML-based interfaces can be a big plus for interoperability and data validation.

- ***Single direction interface:*** JCA 1.0 only defines a single direction call from a client to an EIS application. It doesn't define how an EIS application calls the client back. This makes it difficult to use JCA 1.0 as a specification for process-level integration where bidirection communication is often required.
- ***No adapter metadata:*** JCA 1.0 doesn't have a standard way to query an adapter's metadata.

Is JCA the Right Solution for You?

To answer this question, you need to consider several factors:

- ***Architecture:*** Needless to say, JCA has been widely adopted as an open-industry standard and will flourish in the future. It is architecture neutral, and cheaper and easier to implement than most proprietary solutions. It's simply a better architecture than its competitors.
- ***Availability:*** At this point, JCA adapters are in their early releases for a limited number of EIS applications. If there's a solution for your application server and EIS application and if it's robust enough for your application, you may want to give it a try. Otherwise, webMethods' EIS integration solution has been around for several years and is a safer bet. In addition, JCA adapters from different vendors may have different non-JCA features, such as bidirectional communication, asynchronous message delivery, and an XML-based interface, that can be critical factors in your decision.
- ***Application arena:*** As I discussed earlier, JCA is currently performance-optimized for synchronous connectivity and designed to benefit from the QoS provided by the J2EE application server. It's a great fit for an EAI application, but not for B2B integration.

Whether the JCA is for you should be decided on a case-by-case basis. It's important to keep in mind what the strength and weakness of JCA adapters and their counterparts are and apply your solution accordingly. Whether you choose to use JCA or not, you can learn something from the spirit of JCA architecture. Isolate the callable interface and QoS so that you always have room to migrate to JCA in the future.

AUTHOR BIO

David Li is the president of VertFuture Inc (www.vertfuture.com), a consulting firm specializing in B2B and EAI development and integration. He holds a PhD in chemical physics from the University of Pennsylvania.

dli@vertfuture.com

Editor's Note

The Java Connector Architecture completes the story for building enterprise applications for J2EE. Before J2EE CA was available, J2EE application servers enabled development of all the pieces of the jigsaw puzzle except the piece that connected to the legacy systems the data was extracted from. To integrate with these EIS systems, companies provided custom development for each system.

With J2EE CA, a layer of abstraction is provided by the application server vendors for connecting Java platform components to back-end systems. This also fits into the natural evolution of the J2EE application server market. Because of the popularity of J2EE, EIS vendors had to follow suit and offer resource adapters to connect to J2EE components.

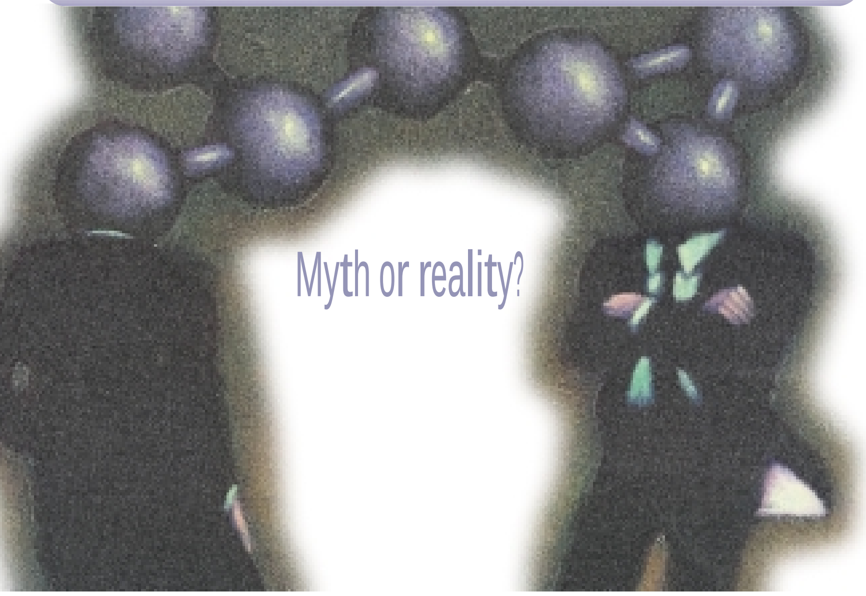
At some point the adapter framework and the corresponding adapters should become a commodity offering from J2EE app server vendors. Visualize a design studio, similar to the one offered by BizTalk, where basic connectivity to ERP systems is available in a graphical environment. Of course, customization will still be required, depending on the needs of specific clients. However, getting a prototype up and running shouldn't be as difficult as it is today.

silverstream software

www.silverstream.com/challenge

End-to-End Transactionality

Myth or reality?



WRITTEN BY
MARK C. LITTLE

Recently, the question was asked whether or not the models on which current transaction systems are based (e.g., JTA and JTS) are powerful enough to support end-to-end transactional guarantees for applications.

In addition, it was suggested that traditional Online Transaction Processing systems (OLTP) don't suffer from such limitations, rendering them more suitable for the emerging e-commerce applications that may require such guarantees.

This article discusses this question and shows that there's nothing inherently wrong with these new models that prevents applications from using them to obtain end-to-end transactionality. However, before addressing the question of whether or not any specific transaction system can be used to provide end-to-end transactional guarantees, it's important to realize the following: end-to-end transactionality is not some holy grail that people have been searching for in myths and legends; it's a solution to one specific problem area, not a global panacea to all transaction issues. The ability or lack thereof to guarantee end-to-end transaction integrity does not in and of itself prevent a specific transaction system provider from tackling many other equally important issues in today's evolving world of e-commerce and mobile applications.

AUTHOR BIO
Dr. Mark Little is an engineer/architect for HP Arjuna Labs in Newcastle upon Tyne, England, where he leads the HP-TS and HP-WST teams. He is HP's representative on the OTS Revision Task Force and the OASIS Business Transactions Protocol specification.

What Is End-to-End Transactionality?

Let's consider what such end-to-end guarantees are. Atomic transactions (transactions) are used in application programs to control the manipulation of persistent (long-lived) objects. Transactions have the following ACID properties:

- ***Atomic:*** If interrupted by failure, all effects are undone (rolled back).
- ***Consistent:*** The effects of a transaction preserve invariant properties.
- ***Isolated:*** A transaction's intermediate states are not visible to other transactions. Transactions appear to execute serially, even if they're performed concurrently.
- ***Durable:*** The effects of a completed transaction are persistent; they're never lost (except in a catastrophic failure).

A transaction can be terminated in two ways: committed or aborted (rolled back). When a transaction is committed, all changes made within it are made durable (forced on to stable storage, e.g., disk). When a transaction is aborted, all the changes are undone. Atomic transactions can also be nested; the effects of a nested transaction are provisional upon the commit/abort of the outermost (top-level) atomic transaction.

Commit Protocol

A two-phase commit protocol is required to guarantee that all the transaction participants either commit or abort any changes made. Figure 1 illustrates the main aspects of the commit protocol: during phase one, the transaction coordinator, C, attempts to communicate with all the transaction participants, A and B, to determine whether they'll commit or abort. An abort reply from any participant acts as a veto, causing the entire transaction to abort. Based upon these (lack of) responses, the coordinator decides whether to commit or abort the transaction. If the transaction will commit, the coordinator records this decision on stable storage and the protocol enters phase two, where the coordinator forces the participants to carry out the decision. The coordinator also informs the participants if the transaction aborts.

When each participant receives the coordinator's phase-one message, they record sufficient information on stable storage to either commit or abort changes made during the transaction. After returning the phase-one response, each participant that returned a commit response must remain blocked until it has received the coordinator's phase-two message. Until they receive this message, these resources are unavailable for use by other transactions. If the coordinator fails before this message is delivered, these resources remain blocked. However, if crashed machines eventually recover, crash recovery mechanisms can be employed to unblock the protocol and terminate the transaction.

A participant's role depends on the application in which it occurs. For example, a J2EE JTA XAResource is a participant that typically controls the fate of work performed on a database (e.g., Oracle) within the scope of a specific transaction.

Note: The two-phase commit protocol that most transaction systems use is not client/server-based. It simply talks about a coordinator and participants, and makes no assumption about where they're located. Different implementations of the protocol may impose certain restrictions about locality, but these are purely implementation choices.

What Is an End-to-End Transaction?

In this section I use the Web as an example of where end-to-end transacspiritsoft

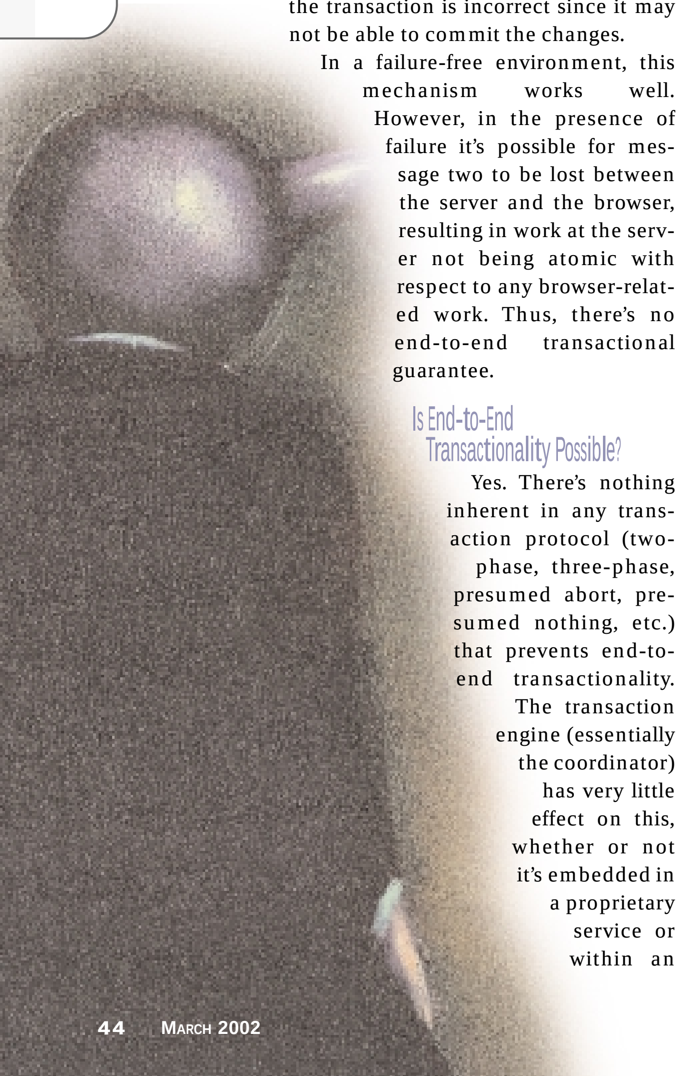
www.spiritsoft.net/climber

tionality integrity is required. However, there are other areas (e.g., mobile) that are equally valid. Transposing the issues mentioned here to these other areas should be relatively straightforward.

Atomic transactions, with their "all-or-nothing" property, are a well-known technique for guaranteeing application consistency in the presence of failures. Although Web applications exist that offer transactional guarantees to users, these guarantees extend only to resources used at or between Web servers; clients (browsers) are not included, even though they play a significant role in certain applications, as mentioned earlier.

Therefore, providing end-to-end transactional integrity between the browser and the application (server) is important, as it allows work that involves both the browser and the server to be atomic. However, current techniques based on CGI scripts can't provide end-to-end guarantees. As shown in Figure 2, the user selects a URL that references a CGI script on a Web server (message one), which then performs the transaction and returns a response to the browser (message two) after the transaction is complete. Returning the message during the transaction is incorrect since it may not be able to commit the changes.

In a failure-free environment, this mechanism works well. However, in the presence of failure it's possible for message two to be lost between the server and the browser, resulting in work at the server not being atomic with respect to any browser-related work. Thus, there's no end-to-end transactional guarantee.



Is End-to-End Transactionality Possible?

Yes. There's nothing inherent in any transaction protocol (two-phase, three-phase, presumed abort, presumed nothing, etc.) that prevents end-to-end transactionality. The transaction engine (essentially the coordinator) has very little effect on this, whether or not it's embedded in a proprietary service or within an industry standard application server. What does make end-to-end transactionality difficult is that it requires a transactional participant to reside at each "end," but it does not require a transaction coordinator and its associated baggage to reside at each "end."

A contract exists between transaction coordinator and participants, which states (in brief and with many simplifications):

- Once the transaction coordinator has decided to commit or roll back the transaction, it guarantees delivery of this information to every participant, regardless of failures. *Note:* There are various optimizations to this, such as a presumed abort protocol, but in essence the contract remains the same.
- Once told to prepare, a participant will make sufficient durable information for it to either commit or cancel the work that it controls. Until it determines the final outcome, it should neither commit nor cancel the work itself. If a failure occurs, or the final outcome of the transaction is slow in arriving, the resource can typically communicate with the coordinator to determine the current progress of the transaction.

In most transaction systems the majority of the effort goes into designing and developing the transaction-coordinator engine, making it as performant and reliable as possible. However, this by itself is insufficient to provide a usable system: participants are obviously required. Although any contract-conformant participant implementation can be plugged into the two-phase protocol, typically the only ones that most people use are those that control work performed on (distributed) databases, e.g., the aforementioned XAResource. This tends to result in the fact that many people equate transactions with databases only, and hence the significant amount of resources required for these participant implementations. However, this is not the case: a participant can be as resource hungry as necessary in order to fulfill the contract. Thus, a participant could use a local file system to make state information durable, for example, or it could use nonvolatile RAM. It depends on what the programmer deems necessary.

With the advent of Java it's possible to empower thin (resource scarce) clients (e.g., browsers) so they can fully participate within transactional applications. Transaction participants tailored to the application and environment where they'll work can be downloaded (on demand) to the client to ensure that they can fully participate within the two-phase commit protocol. There's nothing special about specific transaction system implementations that makes them more easily adapted to this kind of environment. The specialization comes from the end-point resource – the client-side participant.

OLTP vs OO-TP

Are "traditional" online transaction processing (OLTP) engines more suited to end-to-end transactionality guarantees than "newer" object-oriented transaction systems? The quick answer is no. Why should they be? It doesn't matter whether a transaction system is supported by a proprietary remote procedure call (RPC) and stub generation techniques or by an open-standard remote object invocation mechanism such as a CORBA ORB; once the distributed layers are removed, they all share the same core – a two-phase commit protocol engine that supports durable storage and failure recovery. How that engine is invoked, and how it invokes its participants, is immaterial to its overall workings.

The real benefit of OO-TP over OLTP is openness. Over the past eight years there's been a significant move away from proprietary transaction processing systems and their support infrastructure for open standards. This move has been driven by users who traditionally found it extremely difficult to move applications from one vendor's product to another, or even between different versions of a product from the same vendor. The OMG pioneered this approach with the Object Transaction Service (OTS) in 1995, when IBM, HP, Digital, and others got together to provide a means whereby their existing products could essentially be wrapped in a standard veneer; this approach allowed applications developed with this veneer to be ported from one implementation to another, and for different implementations to interact (something else that was extremely difficult to do reliably).

Therefore, it's inaccurate to conclude that OLTP systems are superior in any way to OO-TP equivalents because of their architecture, support environment, or distribution paradigm. OLTP systems are typically monolithic closed systems, tying users to vendor-specific choices for implementations, languages, etc. If the experiences gained by the developers of efficient and reliable implementations of OLTP are transposed to OO-TP, then there's nothing to prevent such an OO-TP system from competing well. The advantages should

lutris
technologies,
inc.

www.lutris.com/javaone

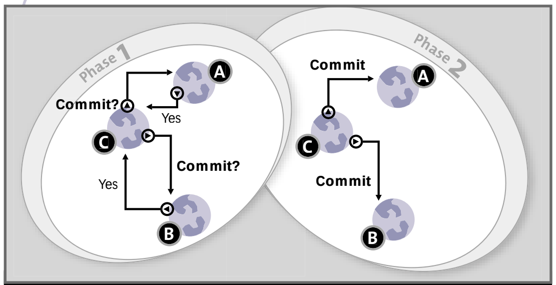


FIGURE 1 Two-phase commit protocol

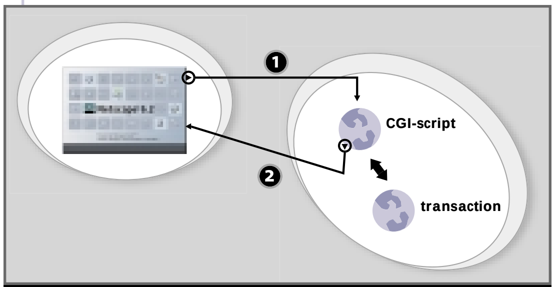


FIGURE 2 Transactions through CGI scripts

be obvious: open systems allow customers to pick and choose the components they require to develop their applications without worrying about vendor lock-in. Such systems are also more readily ported to new hardware and operating systems, allowing customers even more choice for deployment.

The OTS

The OTS architecture provides standard interfaces to components that are possessed by all transaction engines. It doesn't modify the model underlying all the existing different transaction monitor implementations; it mandates a two-phase commit protocol with presumed abort and all implementations of the OTS must comply with this. It was intended as an adjunct to these systems, not as a replacement.

No company that has spent many years building up reliability in such a critical piece of software as transactions would be prepared to start from scratch and implement again. In addition, no user of such reliable transaction software would be prepared to take the risk of transitioning to this new software, even if it were "open." In the area of transactions, which are critical fault-tolerance components, it takes time to convince customers that new technology is stable and performant enough to replace what they've been using for many years.

Although the CORBA model is typically discussed in terms of client/server architecture, from the outset its designers did not want to impose any restrictions on the type of environment in which it could run. There's no assumption about how "thin" a client is, or how "fat" a server must be, in order to execute a CORBA application. Many programmers these days simply use the client/server model as a convenient way in which to reason about distributed applications. But at their core these applications never have what would traditionally be considered a thin client; services that a user requires may well be colocated with that user within the same process. CORBA was the first open architecture to support the configurable deployment of services in this way, correctly seeing this separation of client and service as just that: a deployment issue. Nothing in the CORBA model requires a client to be thin and functionally deficient.

The OTS is comparable to CICS, Tuxedo, and DEC ACMS. It differs only in that it's a standard and allows interoperation. There's nothing fundamentally wrong with the OTS architecture that prevents it from being used in an end-to-end manner. The OTS supports end-to-end transactionality in exactly the same way CICS or any other "traditional" OLTP would – through its resources.

To ORB or Not to ORB?

There also appears to be some confusion as to whether CORBA implemen-

int, inc.

www.int.com

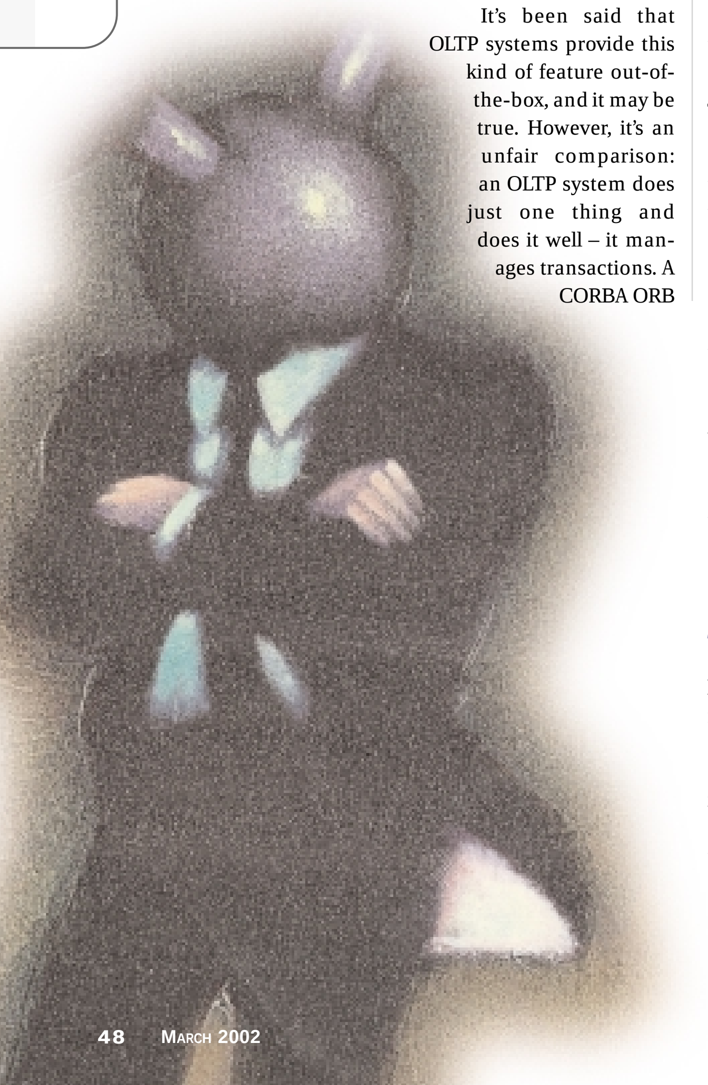
rational
software

www.rational.com/offer/javacd2

tations (ORBs) are sufficient for mission-critical applications. In the mid-'90s when implementations first appeared on the market, their performance was not as good as handcrafted solutions. However, that has certainly changed over the past few years. The footprint of some ORBs is certainly large, but there are other ORBs that have been tailored specifically for real-time or embedded use. Companies such as IBM, IONA, and BEA have seen ORBs develop over the years to become a critical part of the infrastructure that they have to control and therefore they have their own implementations. Other companies have licensed ORB implementations from elsewhere.

The crash failure of an ORB doesn't typically mean that it can recover automatically and continue applications from where it left off. This is because the ORB doesn't have necessary semantic and syntactic information to automatically check point state for recovery purposes. However, by using suitably defined services such as the OTS and the persistence service, it's possible for applications to do this themselves or for vendors to use these services to do this for applications.

It's been said that OLTP systems provide this kind of feature out-of-the-box, and it may be true. However, it's an unfair comparison: an OLTP system does just one thing and does it well – it manages transactions. A CORBA ORB is meant to provide support for arbitrary distributed applications, the majority of which probably won't even need fault tolerance, let alone transactions. However, for those applications that do need these capabilities, it's entirely possible to provide exactly the same recovery functionality using OMG open standards.



J2EE

Although the J2EE model is client /server based, as with CORBA there's nothing to prevent a client from being rich in functionality. It's a deployment choice that's made at build-time and run-time (obviously, the capability for being so rich is required to be built into the client, and even if it were present, it would be up to the user to determine whether or not such functionality was required or possible). *Note:* Although J2EE didn't start out as an infrastructure that used CORBA, it quickly became evident that the OMG companies' experiences in developing CORBA were extremely important to any distributed system. As a result, over the past few years J2EE has gotten closer and closer to the CORBA world, and now requires some critical ORB components in order to run.

The typical way in which J2EE programmers use transactions is through the Java Transaction API (JTA), which is a mandated part of the specification. The JTA is intended as a higher-level API for programmers to try to isolate them from some of the more complex (and sometimes esoteric) aspects of constructing transactional applications. The JTA does not imply a specific underlying implementation for a transaction system, so it could be layered on CICS, Tuxedo, etc. However, because the OTS is now the transaction standard for most companies and it allows interoperation between different implementations, it was decided that the preferred implementation would be based on this (called the JTS, just to place it firmly in the Java domain). The JTS is currently optional, but it may eventually become a mandated part of the J2EE specification.

Application Server Means Thin Client?

No. As shown above, this is essentially a deployment issue. It's certainly correct to say that most J2EE programmers currently use a thin(-ish) client, with most of the business logic residing within the server; however, this is simply because this solution matches 90% of the problems. Closer examination of all application server applications would certainly reveal that although thin clients are the norm, they are by no means the complete picture.

The application server has nothing fundamentally wrong with its model either. Not to say that the client-side of an application server application has to be wafer-thin. If the client wants to embed functionality such as a two-phase aware transactional resource within itself, that's entirely possible. In fact, a client could just as easily be embedded within an application server, if the footprint allowed. The reasons for not doing this have more to do with the footprint size than any architectural issue.

Conclusion

Can end-to-end transactional guarantees be provided by modern transaction systems such as JTA? Yes, as we've shown there's nothing inherent in these models that prevents them from providing such guarantees. The two-phase commit protocol doesn't know anything about clients or servers, doesn't make assumptions about the locality of the coordinator or participants, and doesn't require any semantic knowledge of the applications. End-to-end transactional guarantees are simply a deployment view on the relative locality of different participants.

Are OLTP systems more suited to end-to-end guarantees than their modern OO-TP cousins? As we've shown, since they're both based on two-phase commit protocols, there's nothing in either model that would mean they are any more or less ideal for any specific problem domain. However, there are obvious design and implementation decisions that can be made when building a transaction system using either model, which may mean that specific instances are not best suited for end-to-end transactional solutions. It's important to realize that this is an implementation choice only.

References

- Little, M.C., Shrivastava, S.K., Caughey,S.J. and Ingham, D.B. (1997)."Constructing Reliable Web Applications Using Atomic Actions." Proceedings of the Sixth Web Conference. April.
- Little, M.C. (1997). "Providing End-to-End Transactional Web Applications Using the Object Transaction Service" OMG Success Story.
- "CORBAservices: Common Object Services Specification." OMG Document Number 95-3-31. March 1995.
- "Java Transaction API 1.0.1 (JTA)." Sun Microsystems. April 1999.
- "Java Transaction Service 1.0 (JTS)." Sun Microsystems. December 1999.

mark_little@hp.com

parasoft corporation

www.parasoft.com/jdj3

WRITTEN BY AJIT SAGAR

It's a Brave New World Out There

I've been using the new EJB 2.0 features in my current project and am quite impressed by the much-needed ones that were added to the arsenal of APIs. In addition, the application server vendors have been very quick to provide implementations of the latest and greatest. This confirms that the middle tier of the Java platform is definitely a part of the big league.

Actually if you look at the big league, the players in the middle-tier turf are already well defined – .NET and J2EE. The component models have well-established niches. On the surface, it may look like .NET is more mature, as COM/DCOM has been around for a long time. But .NET is a whole new paradigm, while the J2EE model has been the crux of the Java platform from the get-go.

However, this is not a discussion on .NET versus J2EE; you'll find several discussions on this topic in conferences, on the Web, and in other sources. In this article, I'll examine the repercussions of the J2EE evolution on the world of application servers.

As you probably know, an application server, by definition, is a computer server that serves applications. More precisely, it serves up application services. Its main purpose is to reduce the workload of applications by taking over the mundane activities involved in executing the application and making the application's services available to external modules in a reliable manner. I'll take a stab at defining an application server as follows:

- An application server is a computer program that resides on a server in a distributed network. Its main function is to provide the business logic for an application program.
- An application server provides a customizable and flexible execution environment for hosting business logic components, thus providing distributed services and integrity for application execution.
- An application server provides an execution environment that decouples front-end clients from back-end data access. The execution environment is supported by an infrastructure that enables integration among different applications. Application servers enable this integration by offering software components that can be used to create business logic for an enterprise application. The supporting infrastructure may include architectural frameworks such as messaging systems, transactional managers, and database accessors.

Now let's map this to the J2EE context. Java application servers are a by-product of J2EE's claim on the middle-tier turf. Some of the basic ingredients of J2EE application servers are:

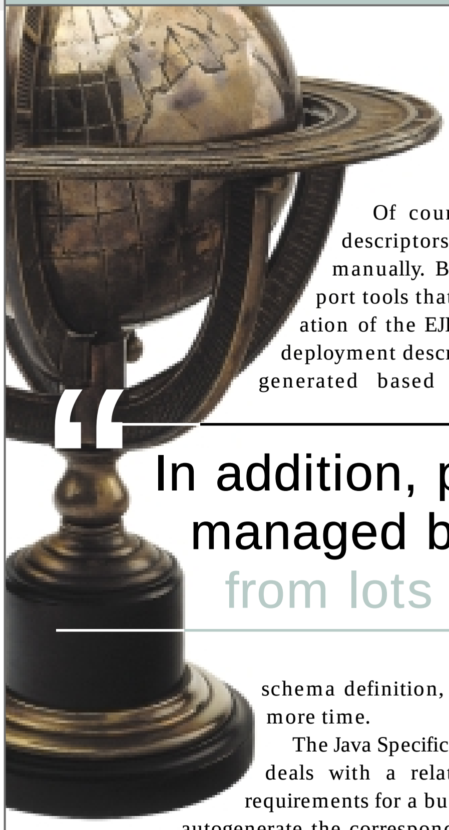
- A standards object model (EJB) for designing business objects
- Uniform APIs for accessing business objects (remote interfaces via RMI)
- Container APIs for interacting with vendors' mechanisms to access system resources (EJB home interfaces)
- APIs for finding business objects (JNDI)
- Standard means of accessing these components through a distributed protocol (Java servlets, RMI)
- Standard APIs for connecting to back-office data sources (JMS/JavaMail, JDBC, JTS)
- Secure access and data interchange (Java security API)
- A standard framework for connecting to legacy systems using adapters to different Enterprise Information System (EIS) resources (Java 2 Connector Architecture)

In the last couple of years, the standards around the J2EE APIs have solidified and been accepted by folks building real-world enterprise applications. This is because many of the missing pieces of the jigsaw puzzle that make up the Java platform have fallen into place. I'd like to take a specific example in EJB 2.0 that illustrates the types of advances made in the last year.

As mentioned earlier, in EJB 2.0 the features have been greatly enhanced. In the previous version (1.1) any serious application needed to use bean-managed persistence for any real access to the data sources. With the current version, there's a big case to be made for using the container-managed persistence. The methods are generated by the container, a requirement for any EJB-compliant J2EE application server. In addition, persistence of table relationships is also managed by the container, saving the developer from lots of tedious code development.

mongoose technology

www.portalstudio.com



Of course, the deployment descriptors still must be written manually. But several IDEs support tools that allow for autogeneration of the EJB classes. Now, if the deployment descriptors could be autogenerated based on some standard schema definition, this would save a lot more time.

The Java Specification Request (JSR) 26 deals with a related issue. To define requirements for a business application and autogenerate the corresponding EJB components, you need to provide the mapping between design elements and software components. To describe software implemented with EJB, UML models must represent EJBs, capturing their structure and semantics. Since UML predates the EJB architecture, it doesn't contain model elements that express the structure and semantics of EJBs. UML was designed to be extensible, however, and provides standard extension mechanisms for defining new model elements. These mechanisms can be used to define new model elements that represent EJBs.

However, there's still a lot of work to be done in this arena. The specifications need to be standardized so that tools and frameworks from different vendors can interoperate. Until this happens, vendors will use nonstandard proprietary mappings between UML and EJBs; some will use nonstandard proprietary metamodels rather than UML, and others won't support modeling or model-based reflection and automation for EJBs.

The realms of the different flavors of application server vendors are also coming together. One primary example is that of Java and XML. XML is a part of most distributed Java applications. Hence, the app server vendors need to provide functionality that steps beyond their traditional realms. The XML IDEs now offer functionality for creating J2EE-based software components, and the J2EE app server vendors offer tools to parse and process XML. Those that can't offer both are integrating with complementary vendors. Soon the lines will become blurred; indeed, this is already happening. In addition, since XML is an enabler across the two major platforms (.NET and J2EE), the vendors will eventually need clean stories around offerings that support both camps.

The development of Web services is another example of an environment that requires integration between tools from different vendors. Ultimately all these environments must translate to platforms that support the modeling of business process flows. These transcend software implementation environments and provide the glue to hook subsystems together. However, the actual translation of business processes to software components is still a daunting task.

As the app server vendors offer more and more sophisticated tools, a question comes to mind. What's left for developers to do if all the software is automatically written for them? Or if the tools become so sophisticated that they even generate the presentation layer for them? Well, for one thing, more developers will need to work inside the IDE shops to make this happen. This will suit the technologists who want to work at the root of the problem. The ones who are building business applications can concentrate on problems at the level of the business domain.

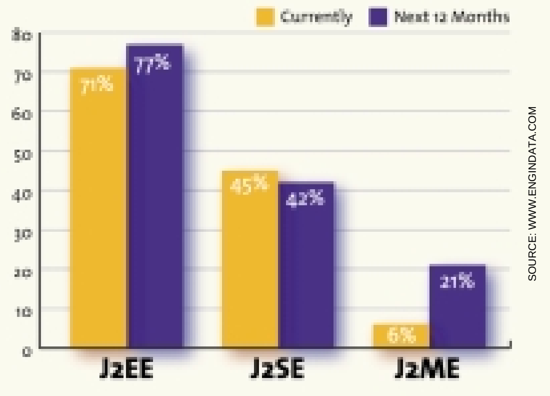
It requires a good understanding of both worlds to create the optimal solution. Software architects need to evaluate and select the features of the IDEs that apply to their particular software and business environment and to direct the development teams to integrate between the two worlds – the world that provides the next level of abstraction on software components and the world that translates business requirements into appropriate components.

ajit@sys-con.com

"In addition, persistence of table relationships is also managed by the container, saving the developer from lots of tedious code development"

Developer Market Survey Report...

Q) Which one of the following Java environments do you develop on and deploy to?



J2EE will see a modest gain of 9% in the next 12 months in development and deployment, while J2SE will actually see a decrease (-7%).

Results were tabulated by EnginData© from more than 10,000 Java developers.

engindata RESEARCH

KEITH BROWN J2SE EDITOR

The IDEal Way Forward for Software Development

My mother bought a computer for her birthday, the usual affair – Windows, printer, scanner, speakers, etc. She's a complete novice and needless to say, she's having a hard time working the thing. Her main complaint (I think in relation to word processing) is that it does far too many things that she doesn't want it to do and the terminology is confusing.

I can empathize – as a Java programmer, Integrated Development Environments (IDEs) have, in the past, confounded me in the exact same way, getting in the way of completing the task. These days I'm using a simple text editor with text highlighting for Java programs and Ant, Apache's Java-based build tool.

I find it much quicker to develop but having said that, I realize there are those who take the time to learn how to "drive" one of these IDEs and swear by them. Maybe I'm just lazy, but I never invested the time and effort into learning the ways of an IDE.

James Gosling recently recognized the fact that there are indeed various levels of developers: people who are not experts at writing code and, presumably, those who are. This is a true observation and IDEs are no doubt a godsend to those who are not experts.

What do the expert code writers think? Do they use these beasts? Often I hear the mumblings, mockings, and dismissals of hard-core engineers as they berate the way of the wizard and gasp in horror at the underlying code it produces. There's a certain machismo in using a plain text editor to develop software.

Perhaps IDEs are just another step in the journey from the machine languages that Alan Turin (or perhaps Mr. Babbage himself) would have rattled out, and the so-called ease of use of 4G languages and the world of wizards. We don't have any programming languages that use natural language yet, and if it were possible, would it be desirable?

It seems the entire progress of computer languages is based on the assumption that they should become more like everyday languages (usually English, as fate would have it) and yet it is developers who complain the most about the wizard approach to programming.

Who's driving this progress? If "hard-core" developers don't approve of this approach, then who's behind the demand for tools such as IDEs? Where does Java fit into all of this? Wasn't Java designed to be easier to learn than its predecessors by extracting a lot of the nuts and bolts away from the developer and utilizing an object-based system?

Industry drives the demand. Industry has a commercial need to utilize the latest in information technology. If this can be made easier and allow more people to become practitioners of languages such as Java, then so much the better – they can have that internal reporting tool up and running in no time at all.

There's an unnerving contradiction going on here somewhere but I can't quite put my finger on it. If computer languages are to progress in the same way as they have been, with the honorable goal of expanding the number of programmers, then this could be a problem for developers.

If universities are to continue to churn out conversion course graduates with nine months of experience, will they be as able as the computer scientists? Probably not, but then isn't that what high-level languages are all about? Making it quicker and easier to learn how to build software.

I seem to have asked more questions than I've answered! Oh well! Like I said, I can't quite put my finger on it.

I'm glad I didn't get an IDE for my birthday because I would probably be pulling my hair out just like my mother is. How many menus can one piece of software have!

keith.brown@sys-con.com

AUTHOR BIO

Keith Brown has been involved with Java for many years. When he's not coding up client solutions for a European Java company, he can be found lurking in the corridors of conferences all around the world.

J2SE INDEX

The IDEal Way Forward for Software Development
It seems the entire progress of computer languages is based on the assumption that they should become more like everyday languages (usually English, as fate would have it)...
by Keith Brown 54

JavaCC
A powerful tool enhances developer productivity
by Sriram Sankar 56

Ask Dr. Java
A source for your Java-related questions
by James McGovern 66

An API Developer's Primer
What's hot and what's not
by Richard Deadman 72

Extending the J2SE 1.4 Logging Package
Monitoring made easy
by Jim Mangione 82

Mac OS X and Java
A revolutionary step toward usability
by Hitesh Seth 88

insession technology

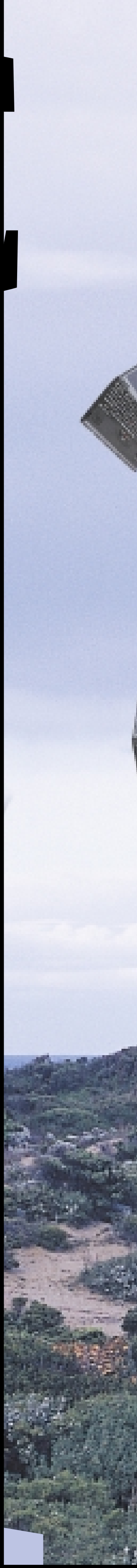
www.insession.com

JavaCC

The Evolution of NewWave Parser Generation Technology

Powerful tool enhances developer productivity

A parser is a program that takes a file as input and determines whether or not it adheres to a prescribed syntax. It's difficult to write a good parser from scratch. However, it's quite easy to write the syntax that it's supposed to check against.



The technology to automatically generate a parser from this syntax specification has existed for around 20 years and is now mature enough to use in a product setting. A parser generator is a software program that accepts a syntax specification as input and generates a parser for that syntax as output.

JavaCC (Java Compiler Compiler) is a parser generator that embodies the state of the art in parser generation technology and generates parsers in Java. Sreeni Viswanadha and I jointly developed JavaCC while we were at Sun Microsystems. Robert Duncan also made some important contributions. Although JavaCC is owned by Sun Microsystems and Sreeni and I are currently at WebGain, we continue to maintain JavaCC. JavaCC is available for a free download from WebGain's Web site, as well as from Sun Microsystems.

The use of parsers and parser generators is ubiquitous. There's always a need for parsers in any large software system. Products such as WebLogic, SQL/JDBC, JPython, Cloudscape, Velocity, BeanShell, JFactor, and Dynamo all use parsers generated by JavaCC or its main competitor – ANTLR.

Especially now, with so many text processing applications being built, it's quite valuable to possess the skills to use parser generators – the time saved by using a parser generator can be immense. As an extreme example, I recently had to write an application that extracted all the type names from a Java program. What took a couple of hours to do with JavaCC would've taken a month to do from scratch.

This article provides a background on parser generation technology, a summary of how JavaCC evolved, and a tutorial on its use.

Parsing, Semanticization, et al.

The processing of files typically requires parsing them and then performing some additional work. The term *parsing* is used differently by different groups, so I'd like to precisely define this term through examples using a Java compiler.

Consider the following Java class:

```
class Interval_1 {
 int left, right;
}
```

This is a correct Java class, which is accepted by the Java compiler, and bytecode will be produced. Let's modify this class as follows:

```
class Interval_2 {
 nosuchtype left, right;
}
```

This is no longer correct because a type name is used (nosuchtype) that doesn't correspond to any declared type. However, this class is syntactically correct (i.e., it conforms to the syntax of the Java language). Let's make another modification:

```
class Interval_3 {
 int left; right;
}
```

This isn't a correct Java class either because it uses a semicolon where it should have used a comma. This class is wrong in a manner different from that of Interval_2 in that it isn't syntactically correct.

Parsing catches errors such as the one in Interval_3. Parsing won't catch the error in Interval_2. To catch that error, processing beyond parsing is required. This requires analyzing the data structures produced by the parsing process. The term *semanticization* is used to describe the analysis that takes place after parsing on programs to catch errors such as in Interval_2.

Typically, parsing can detect errors that can be described by a grammar for the language. Depending on the sophistication of the parser and the complexity of the grammar, the delineation of responsibility between parsing and semanticization can vary.

The Evolution of Parsing Technologies

In the early days of language processing, parsers were written by hand. Typically, the syntactic (parser) checks and the semantic checks were all mixed together in one action, leading to highly complex code that was very difficult to maintain. In the '60s and early '70s research in computer language theory led to various important categorizations and algorithms that have since permitted the automation of building parsers. Three of the important language categories are described below, from most complex to simplest.

- ***Turing languages/Turing machines:*** The Turing machine is an abstraction of a computer program. Every computer program has a corresponding Turing machine. Turing languages are those that can be recognized by Turing machines. For the purposes of this article, this definition is mainly theoretical; it simply provides an upper limit to the complexity of computer languages. All programming languages fit under the umbrella of Turing languages.
- ***Context-free languages:*** Those languages that can be described by grammars, which consist of a set of productions, each describing a particular aspect of the language. You must have seen such grammars in your Java texts. An example of a production is:

```
methodcall ::= name "(" parameterlist ")"
```

This says that a method call starts with a name followed by an open parenthesis, then a list of parameters, and then the closing parenthesis. To complete this definition, other productions must define name and parameterlist. *Note:* Java is not in itself context-free; it's just that a large portion of the language can be described using a context-free grammar. Java language rules such as "A variable must be declared before it can be used" aren't context-free (and can't be described by a grammar). Such rules are processed in the semanticization phase.

- ***Regular expressions:*** The simplest of the three language forms described here, regular expressions can be recognized by finite-state machines. A finite-state machine is a graph where the nodes are the different states that the machine may be in, and the directed edges between the states are the characters allowed as the next character in the input to transition from one state to the other. Regular expressions (and finite-state automata) can be described by a simple grammar (which is less expressive than a context-free grammar). The Java identifier is an example of a regular expression that can be described by the following grammar:

```
identifier ::= firstcharacter othercharacter*
```

This says that an identifier is composed of a first character followed by 0 or more (this is what the asterisk indicates) other characters. To complete this definition, firstcharacter and othercharacter need to be defined.

A main simplification in a regular grammar as compared to a context-free grammar is that the regular grammar can't be recursive (i.e., a production can't use itself directly or indirectly in its description).

A significant step in the evolution of parsing technologies was the development of algorithms and tools that could convert context-free grammars and regular grammars into programs that could check for the correctness of an input with respect to the grammar.

Since then, most computer languages have been designed so that programs are a sequence of tokens, each of which is a regular expression, and the tokens in turn are structured according to a context-free grammar. Not all of the tokens participate in the context-free structure; some are discarded. In the case of Java, discarded tokens include comments and spaces. The following Java program example illustrates this:

```
// This is a simple Java class
class Simple {
 int dummy;
}
```

The tokens in this Java program are listed below:

1. Comment: // This is a simple Java class
2. Reserved word: class
3. Single space
4. Identifier: Simple
5. Single space
6. Delimiter: {
7. Newline
8. Double space
9. Reserved word: int
10. Single space
11. Identifier: dummy
12. Delimiter: ;
13. Newline
14. Delimiter: }

"JavaCC is a parser generator that embodies the state of the art in parser generation technology"

Before the context-free grammar is matched, some tokens are discarded. In this example tokens 1, 3, 5, 7, 8, 10, and 13 are discarded (the discarded tokens are referred to as "white space"), leaving behind the tokens "class", "Simple", "{", "int", "dummy", ";", and "}".

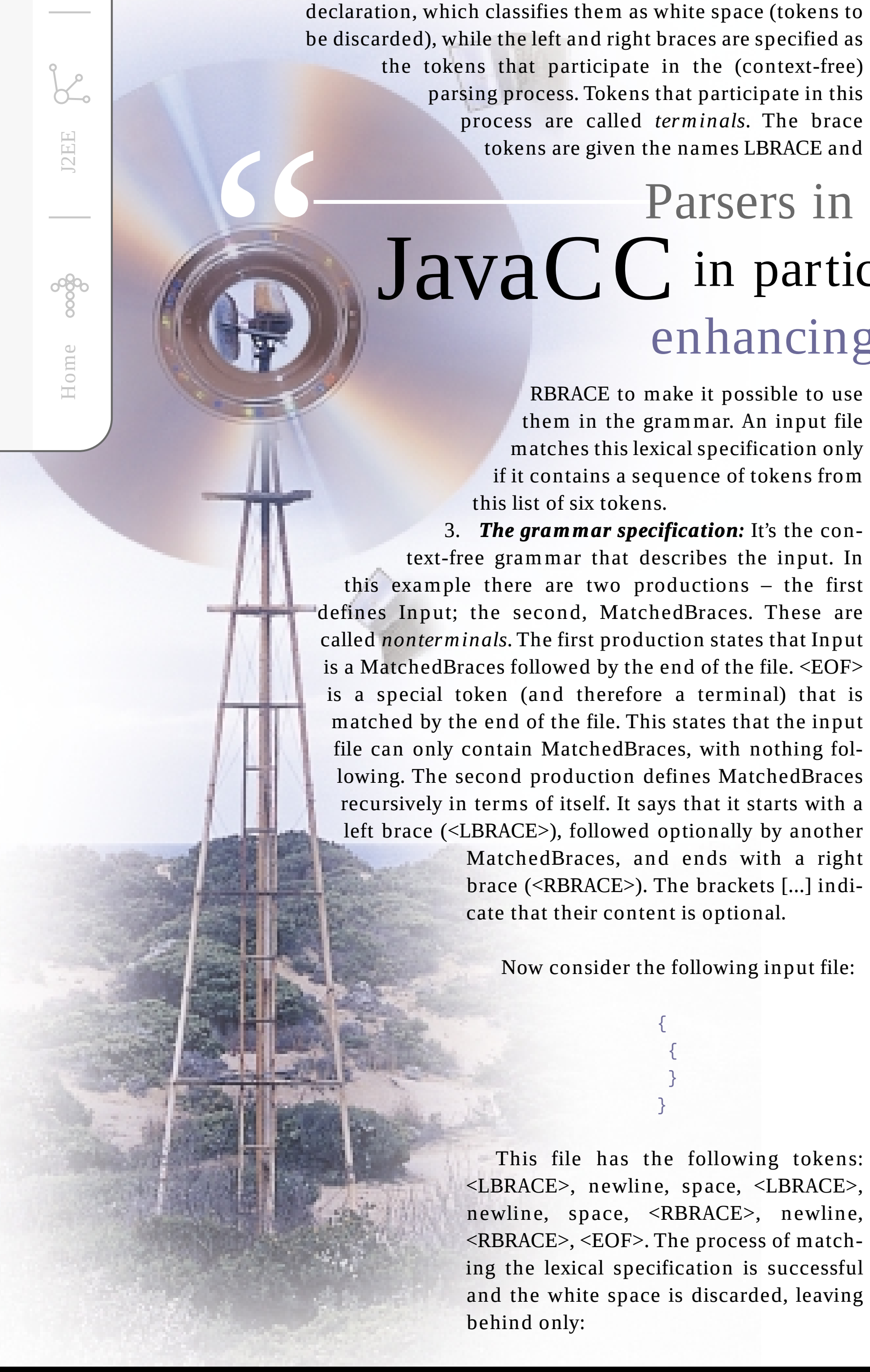
These tokens are then matched with the context-free grammar for Java.

Using JavaCC — A Simple Example

Listing 1 is a simple grammar file written in the JavaCC grammar syntax. It specifies that the input file should contain a sequence of left braces followed by another sequence of matching (equal in number) right braces. The file contains three parts:

john wiley
& sons

wiley.com



1. ***The class specification:*** A Java class into which the parser is generated by JavaCC, this class will have the fields and methods described in the class specification. In addition, a method is generated that corresponds to each production in the grammar. In this example the class specification states that the class is called Simple and it contains a main method that simply creates a parser object (of type Simple) and invokes it on the standard input. The method Input, which is called by the main method, is generated by JavaCC to correspond to one of the productions in the grammar specification.
2. ***The lexical specification:*** Describes the tokens present in the input file using regular expressions. In this example the tokens are quite simple and can therefore be expressed as strings (the simplest form of regular expressions). Six tokens are described: the space, tab, newline, carriage return, left brace, and right brace. The first four are specified in a SKIP declaration, which classifies them as white space (tokens to be discarded), while the left and right braces are specified as the tokens that participate in the (context-free) parsing process. Tokens that participate in this process are called *terminals*. The brace tokens are given the names LBRACE and RBRACE to make it possible to use them in the grammar. An input file matches this lexical specification only if it contains a sequence of tokens from this list of six tokens.
3. ***The grammar specification:*** It's the context-free grammar that describes the input. In this example there are two productions – the first defines Input; the second, MatchedBraces. These are called *nonterminals*. The first production states that Input is a MatchedBraces followed by the end of the file. <EOF> is a special token (and therefore a terminal) that is matched by the end of the file. This states that the input file can only contain MatchedBraces, with nothing following. The second production defines MatchedBraces recursively in terms of itself. It says that it starts with a left brace (<LBRACE>), followed optionally by another MatchedBraces, and ends with a right brace (<RBRACE>). The brackets [...] indicate that their content is optional.

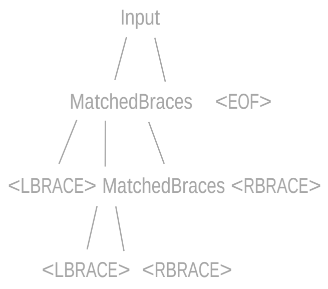
Now consider the following input file:

```
{
 {
 }
}
```

This file has the following tokens: <LBRACE>, newline, space, <LBRACE>, newline, space, <RBRACE>, newline, <RBRACE>, <EOF>. The process of matching the lexical specification is successful and the white space is discarded, leaving behind only:

```
<LBRACE> <LBRACE> <RBRACE> <RBRACE> <EOF>
```

The parsing process then successfully matches these tokens with the grammar as follows:



The diagram shown above is called a *parse tree.*

You can try this example yourself. First create a grammar file with the above contents. Call the file Simple.jj (you can call it anything, but by convention you should give it the jj extension). Download JavaCC from www.webgain.com and install it on your computer. Add the bin directory within this installation into your path. Now you should be able to run "javacc" from your command prompt (DOS window or UNIX xterm). Type:

```
javacc Simple.jj
```

It produces a bunch of Java files as output. Compile these Java files by typing:

```
javac *.java
```

Now create different kinds of input files – both good and bad inputs – and then parse them using the generated parser by typing:

```
java Simple < InputFile
```

At this point you've taken the biggest and most important step toward learning JavaCC. The rest of the learning process will happen very quickly.

“Parsers in general, and JavaCC in particular, are powerful tools for enhancing developer productivity”

Actions

The previous example shows the use of a JavaCC parser in determining whether an input matches its grammar. If it matches, the parser quietly returns; if it fails, the parser prints an error message detailing the reason for failure.

It's also necessary to be able to make the parser do something useful on a successful parse beyond simply declaring success. For this purpose JavaCC provides *actions*, Java code that's inserted at an appropriate location in the grammar and executed as the parser matches that portion of the grammar. Actions may be arbitrarily complex and may invoke library methods. There are two general kinds of actions: lexical and parser.

Lexical actions are executed after a successful match of a token. *Parser* actions are embedded within the grammar

dice.com

www.dice.com

and are executed as the parser goes past a point in the parsing process. The grammar file in Listing 2 extends the previous one with both lexical and parser actions. These actions are simply print statements; try running the parser generated from this grammar file with the inputs you created for the previous exercise to see how the actions are executed.

Once you complete this exercise, you should be able to go through the sequence of examples in the JavaCC download and systematically learn more details about JavaCC. Once you've gone through the examples, you should be ready to embark on any complex parsing application.

Lookahead and Ambiguity Resolution

Both lexical and grammar specifications can be ambiguous. In such situations standard ambiguity resolution schemes take effect.

Consider the lexical specification for the Java language. It defines all the Java reserved words as well as Java identifiers. What should the parser do when it encounters the sequence "interface"? The obvious answer is that it should recognize it as the token corresponding to the reserved word "interface". However, there are other ways to recognize it, such as an identifier "interface" or the reserved word "int" followed by the identifier "erface".

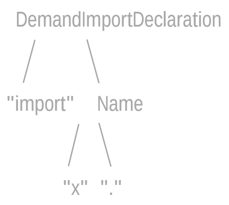
The ambiguity resolution scheme for lexical specifications is that the longest match is preferred (so, in the above example, "interface" is preferred over "int"), and when there are multiple matches of the same length, the match that occurs first in the grammar file is used. In the example above, the reserved word "interface" is selected over the identifier "interface" if the lexical specification for the reserved word occurs earlier than that of identifier.

Ambiguities in grammar specifications are resolved by looking at the next available token and proceeding with the first successful match. This resolution scheme is referred to as using a lookahead of one token. Consider the grammar for demand import declarations in Java.

```
void DemandImportDeclaration() :
{}
{
 "import" Name() "." "*" ";"
}

void Name() :
{}
{
 <IDENTIFIER> ( "." <IDENTIFIER> )*
}
```

On seeing the input "import x.*;", the parser wil match the "." with the one in the production for Name. It will fail the parse because it expects an identifier and not a "*". This grammar is therefore wrong because the ambiguity is resolved incorrectly. The parse tree describing this parse process is shown in the following diagram.



To correct this problem, we must specify that the parser looks ahead more tokens before it decides how to parse. In this example a lookahead of two tokens will work. This can be specified as a global setting (so that a lookahead of two tokens is used for all choices) or a lookahead of two tokens can be specified for this particular point only. The global setting causes the parser to become rather inefficient – typically, most choices can be made correctly with a lookahead of one – so the latter approach is the correct one. The grammar, modified with the additional lookahead specification, is shown below.

```
void DemandImportDeclaration() :
{}
{
 "import" Name() "." "*" ";"
}

void Name() :
{}
{
 <IDENTIFIER> ( LOOKAHEAD(2) "." <IDENTIFIER> )*
}
```

There are other more complex ways to specify lookaheads, but we won't go into them here. You can study the JavaCC documentation and examples for more information.

Advanced Topics

Other aspects of JavaCC not described here but still very important and of significant value to JavaCC are listed below:

- ***Special tokens:*** Sometimes it's preferable to process some white space tokens during the parsing process. JavaCC provides a way to specify tokens so they don't get discarded but still don't participate in the parsing process.
- ***Lexical states:*** This allows you to split your lexical specification into different categories within which matches and ambiguity resolution take place independently of the other states. This allows you to conveniently write lexical specifications for (say) mail files, where the string "From" may be a reserved word in one context and a simple text in another.
- ***Java code productions:*** This allows you to write actual Java code for a production instead of a grammar. Sometimes it isn't possible to write a production using the context-free syntax, and Java code productions become invaluable. However, this feature must be used sparingly.
- ***JJTree:*** This is a JavaCC preprocessor that inserts parse tree building actions into a grammar based on a parse tree specification. The result of the parsing process is a tree data structure that captures the essence of the parsing process.

"JavaCC strongly supports the industry trend toward standardizing the syntax of important data formats such as XML"

oracle corporation

www.oracle.com/hurwitz

Additional processing is then performed on this tree (no additional actions are added by hand into the grammar).

Conclusion

Parsers in general, and JavaCC in particular, are powerful tools for enhancing developer productivity. There's always some need in any large software system for parsers, and the use of parsers and parser generators is ubiquitous.

Like other parser generators, JavaCC reads a grammar specification and converts the code into a Java program that recognizes matches to the grammar. In addition to the parser generator itself, JavaCC also provides other standard capabilities related to parser generation, such as tree building (via a tool called JJTree), actions, and debugging.

JavaCC strongly supports the industry trend toward standardizing the syntax of important data formats such as XML. By giving developers a powerful tool for automating the creation of Java code that can recognize a standardized language structure, JavaCC will play a valuable role in the evolution of open systems.

AUTHOR BIO

Sriram Sankar is VP of engineering at WebGain, a Java developer products company. He came to WebGain through the acquisition of Metamata, where he was founder and CEO. Sriram previously worked for Sun Microsystems and Stanford University. He holds a PhD in computer science from Stanford University.

ssankar@webgain.com

Listing 1

```
// 1. Class specification

PARSER_BEGIN(Simple)

public class Simple {

 public static void main(String args[]) throws ParseException
{
  Simple parser = new Simple(System.in);
  parser.Input();
 }

}

PARSER_END(Simple)

// 2. Lexical specification

SKIP :
{
 " "
| "\t"
| "\n"
| "\r"
}

TOKEN :
{
 <LBRACE: "{">
| <RBRACE: "}">
}

// 3. Grammar specification

void Input() :
{}
{
 MatchedBraces() <EOF>
}

void MatchedBraces() :
{}
{
 <LBRACE> [ MatchedBraces() ] <RBRACE>
}
```

Listing 2

```
// 1. Class specification

PARSER_BEGIN(Simple)

public class Simple {

 public static void main(String args[]) throws ParseException
{
  Simple parser = new Simple(System.in);
  parser.Input();
 }

}

PARSER_END(Simple)

// 2. Lexical specification

SKIP :
{
 " "
| "\t"
| "\n"
| "\r"
}

TOKEN :
{
 <LBRACE: "{"> { System.out.println("Lexical action:
Encountered LBRACE"); }
| <RBRACE: "}"> { System.out.println("Lexical action:
Encountered RBRACE"); }
}

// 3. Grammar specification

void Input() :
{}
{
 MatchedBraces() <EOF>
}

void MatchedBraces() :
{}
{
 <LBRACE>
 { System.out.println("Parser action: Successfully parsed
LBRACE"); }
 [ MatchedBraces() ]
 <RBRACE>
 { System.out.println("Parser action: Successfully parsed
RBRACE"); }
}
```

capella
university

www.capellauniversity.edu

Prescriptions for Your Java Ailments

Ask Doctor Java

WRITTEN BY
JAMES McGOVERN

Unlike the doctor who works for your HMO, I won't require a copayment for each visit nor ask you to fill out long arduous forms. I'm here to help readers of *Java Developer's Journal* find a cure for their Java system ills.

Q What unique parameters accessible from pure Java can be used to identify a machine? We're working on a licensing mechanism for software currently in development. We've toyed with using the IP address, but this won't work with dynamic IP addresses or if there's no network connection. There doesn't seem to be a lot that can be discovered about the machine using only Java, and if there is, it isn't very well documented.

A You can do a ton of things to add licensing features to your product. Let's discuss some of the techniques and the pros and cons of each. Let's first consider the use of dongles within your application. If the user is running Windows 95, 98, or ME, Linux, and so on, you can query the serial port from Java. If your application is running on Windows NT, 2000, or XP, dongles won't work. It's possible to use a USB-based dongle, but this won't be supported by Windows 95 or NT. Of course, depending on the choice of dongle, you'll have to incur hardware costs.

You could query settings within each operating system – provided you want to write custom JNI code for each target platform. For instance, on the Windows operating system you could make a Win32 call to GetSystemInfo, which will return information about the machine, such as the number of processors and their type (e.g., Intel, AMD, Cyrix, and so on). On a Windows platform you could also query the registry to determine the particular machine's GUID (Global Unique ID), which is guaranteed unique by Microsoft. This, of course, would mean the user couldn't move the application to another machine without contacting you. Many platforms support storing unique serial numbers on the CPU chips themselves, but this can be turned on and off in the BIOS.

If you can at least guarantee that the machine has a network card, regardless of whether it's connected to the network, you could use JNI and/or run the netstat command to get its MAC address. The MAC address is also globally unique. If you could guarantee a network connection, you could consider implementing Kerberos, where each time the program starts, it checks with a central server (most likely in your company) for authorization to run.

If you want to explore the Java security packages, you could consider implementing digital certificates. Upon startup, Java could check for the existence of a valid digital certificate. You could investigate becoming your own Root CA or at least use a Test CA to check the certificate against your own keystore.

Finally, trying to implement any of the above recommendations will make the average developer's head hurt. I recommend considering a vendor who's developed easy-to-use APIs you could easily integrate into your product – FlexLM (www.flexlm.com), for example. Many tools, such as Rational Rose, use their licensing approach.

Q I've been tackling EJBs for a while using stateless session beans. Now I want to use entity beans. In all that I've read, the ejbCreate() method in the entity bean returns the primary key of the newly created entity. The caller provides it with the primary key (hence no problem).

I'm using the autoincremented primary keys found in most databases (I'm using MS SQL), so I insert only the record but not the primary key. After the insert I don't want to requery the database to find out the newly created primary key. Is it mandatory that I return the primary key?

A I've got the answer to your problem and it's easier than you may think. The primary key class doesn't need to be the same as the primary key in the database. The only requirement for primary key classes is that they be unique. You can accomplish this by overriding the hashcode() and equals() methods so two rows don't equal.

The trick is to make your EJB's primary key some unique combination of information that isn't related to your autoincrementing sequence number. Let's say you were developing a human resources application and the respective employee bean. You could make the database primary key the employee ID and make the EJB primary key the social security number.

For those who can't create a unique key via hashcode(), you'll have to requery. Of course there may be other scenarios in which you need to consider returning the "right" primary key, especially if your container is aware of the relationships. This solution should work for most applications.

Q I agree with your assessment in a previous issue (*JDJ*, Vol. 6, issue 11) that there's no runtime overhead in using java.util.* instead of java .util.Vector. However, at least in the case of the equivalent construct in Ada ("use" instead of "import"), there could be considerable compile-time overhead involved in using java.util.*.

The reasoning on Ada is analogous to this: suppose the code has import java.util.*; and a whole lot of other import calls as well, then the compiler comes across a line of code like:

```
Vector v = new Vector();
```

Even if the compiler quickly finds Vector in the java.util package, it can't quit there. It has to look through all the other packages to see if any of them also have a Vector class. If they do, it must issue an error message. Otherwise it'll use the Vector from java.util. Since all this checking has to be done for every reference, it can be time-consuming.

I don't know if the same reasoning applies to Java compilers, but it seems it might. I agree with your recommendation to use "single type imports." In fact, for readability's sake, the Ada community consensus

apptricity

www.apptricity.com

was that in many cases you shouldn't use import ("use") at all. If you were doing a lot of math from a math package, then it would be okay to say:

```
import xxx.math.*;
```

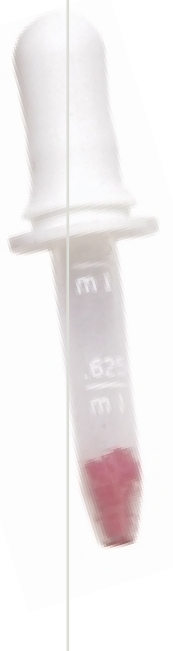
If you're using only one thing from some obscure place, it makes the code more readable to fully qualify where the names come from. But readability is a separate issue.

A I'm at a loss as I've never programmed in Ada. I used to program for Microsoft, and in the past I also programmed in PowerBuilder. I have a couple of thoughts I'd like to share: first, unless you are designing monolithic packages, you likely won't notice the difference in compile times when using imports such as java.util.*. Any design with a lot of import statements in the class file should also be suspect. While the actual bytecode doesn't have references to the import statements, having too many of them will make it difficult for those who need to maintain it after you've moved on to greener pastures.

Many of the Java development environments such as WebGain Studio have code analyzers that will immediately point out unused import statements. These tools should be a part of the quality assurance process of any development project.

Q Java uses several distinct "listener" patterns throughout the language, including observer/observable, AWT event handling, and the beans approach. Which model do you recommend developers use?

A The answer isn't as obvious as it first appears. Let's look at java.util.Observable first. Essentially, it can be used for applications that require "callback" type functionality. The majority of the heavy work is typically implemented within the Observable class. The Observable class has the responsibility of also notifying all observers by calling notifyObservers(). Each Observer class calls its own update() method. The problem with this approach is that Observable is a class, which means you have to subclass. This may not be practical in many situations.



AWT and beans actually use the same approach. Both implement change listeners and fire events as a notification mechanism. In this approach you're not even required to implement a full-fledged bean to utilize the functionality. Events usually implement the java.util.EventListener interface, which doesn't have the same limitations as the class approach.

Q I have a simulated lift control system running through Java and I've run into a problem. I need to implement an emergency stop system. Each lift has its own thread. When one lift needs to be shut down immediately, a variable is changed in the lift class, which is a loop in the threads run() method. All current commands in that loop are handled and then the thread exits normally. It is re-created when the start command is entered.

Some of the lift actions (moving, stopping at floors) make use of the Thread.sleep() method. The main lift thread is paused using this, thus simulating a pause required for the lift operations to be simulated correctly. If an emergency stop is requested, the main thread must interrupt the lift thread if it happens to be "sleeping." However, nothing I've tried works. Is there a better solution?

A I love the English vocabulary. We Americans prefer the term elevator. Anyway, this question referred to using an event listener approach that may be better for your application. Each object could listen for start, stop, and pause events sent by other objects (security guard console, each floor's button, and so on).

If each request for an elevator gets sent to a controller, the controller could use the wait/notify methods on the threads themselves and send all elevators to the ground floor. If you want to control a single elevator, then a better approach may be to "interrupt" the thread by using Thread.interrupt(). Calling interrupt will cause an InterruptedException to occur, which you can handle with a try/catch block to implement the emergency stop function.

Q I would like to know how to use Java with IRC (Internet Relay Chat) to do some programmatic communication. Could you point me in the right direction?

A On the surface this sounds a little dangerous, but I hope you won't use this information to cause harm. If you're looking to be a user and don't require source, then you can check out JIRC (www.jpilot.com). For an open-source IRC client, visit https://sourceforge.net/projects/objircchat/.

One of the more advanced uses of Java and IRC that I've run across to date is the Chattal project (http://sourceforge.net/projects/chattal/). Its goal is the development of an artificially intelligent Internet Relay Chat agent that mimics human interaction and provides an information repository for learned topics.

Q My application has a properties file and I'd like to know if an end user tampers with it. How can I accomplish this?

A You can take a couple of approaches. First you can inspect the date and time the file was last modified, but this may not give you an accurate picture, as the user could've restored the file from backup or some other way. You can always encrypt the file, but there may be situations where you need the information contained within the file to be human readable.

The simplest approach that comes to mind is to calculate a message digest (one-way hash). The following code snippet should point you in the right direction:

```
String filename = "doctorjava.properties";
MessageDigest md = MessageDigest.
     getInstance("MD5");
FileInputStream fis = new FileInputStream(file
     name);
DigestInputStream dis = new DigestInputStream
     (fis, md);
byte[] buffer = new byte[2048];

while (dis.read(buf) > 0)
 ...
byte[] digest = md.digest();
fis.close();
```

By creating and storing a digest along with a file, you can reload it and compare the values. If the digests are different, then you know someone has changed the contents.

Q How do I compile a string containing Java code on the fly?

A Whoa! I've got your prescription. The answer will require the use of reflection, the Sun community source (www.sun.com.tools package), and the writing of your own classloader. I can't include the entire source required to implement this, but I can point you in the right direction.

Essentially, it would require a method that looks similar to:

```
Public stringCompile(String source, String file
     name) throws CompileException
```

Once you set up this method you can, of course, make defensive copies of the strings you have passed in, as well as create the appropriate exception class. The very first thing we need to do with this method is determine the current classpath. Java's default is to load classes from the classpath. In our scenario we want to use a class on the fly. To determine the classpath, we would do the following:

```
ClassPath cp = new ClassPath(System.
     getProperty("java.class.path"));
```

Next, from the sun.com.tools package, we'll need to create the appropriate BatchEnvironment() passing in the classpath and an OutputStream we'll use later. We'll need to creilog

www.ilog.com/jdj/jrules

"AWT and beans actually use the same approach. Both implement change listeners and fire events as a notification mechanism"

ate a new class (let's call it MemoryClassFile) that we'll pass to the BatchEnvironment.parseFile() method. The method call will appear as follows:

```
BatchEnvironment be;
Be.parseFile(new MemoryClassFile(filename +
     ".java", source));
```

Once the source has been compiled, we'll need to get an enumeration of all the classes. We can accomplish this with BatchEnvironment.getClasses(). We'll also need to figure out which classes have already been compiled. We can determine this easily, as they'll be an instance of BinaryClass.

We'll take the OutputStream and convert it to a byte array; then we'll need to create a class (let's call it MemoryClassLoader) that's derived from ClassLoader, as it is an abstract class. We'll use MemoryClassLoader to load the compiled classes. The method defineClass will convert an array of bytes to an instance of the desired class. Your MemoryClassLoader class should look similar to:

```
Static class MemoryClassLoader extends
     ClassLoader {
public class getClassFromBytes(String filename,
     byte[] bytearray) {
Return defineClass(filename, b, 0,
     bytearray.length);
        }
}
```

Finally we'll need to use reflection to determine which classes are available to the application and which methods they implement. I could see using this technique for spinning your own CASE tool or script language. I wish you well.

Conclusion

JDJ readers have been sending me some pretty tough questions. This was the most challenging column to date. I encourage you to keep sending in difficult questions.

I leave you with this quote by Douglas Adams:

"A common mistake that people make when trying to design something completely foolproof is to underestimate the ingenuity of complete fools."

• • •

Send your questions, praise, comments, and admiration to doctorjava@sys-con.com.

• • •

Published letters will be edited for length and clarity. Any reference to third parties or third-party products should not be construed as an endorsement by Doctor Java or Java Developer's Journal.

Author Bio

Doctor Java, aka James McGovern, works as an enterprise architect with Hartford Financial Services (www.thehartford.com), where he focuses on the architecture of high-availability and J2EE-based solutions.

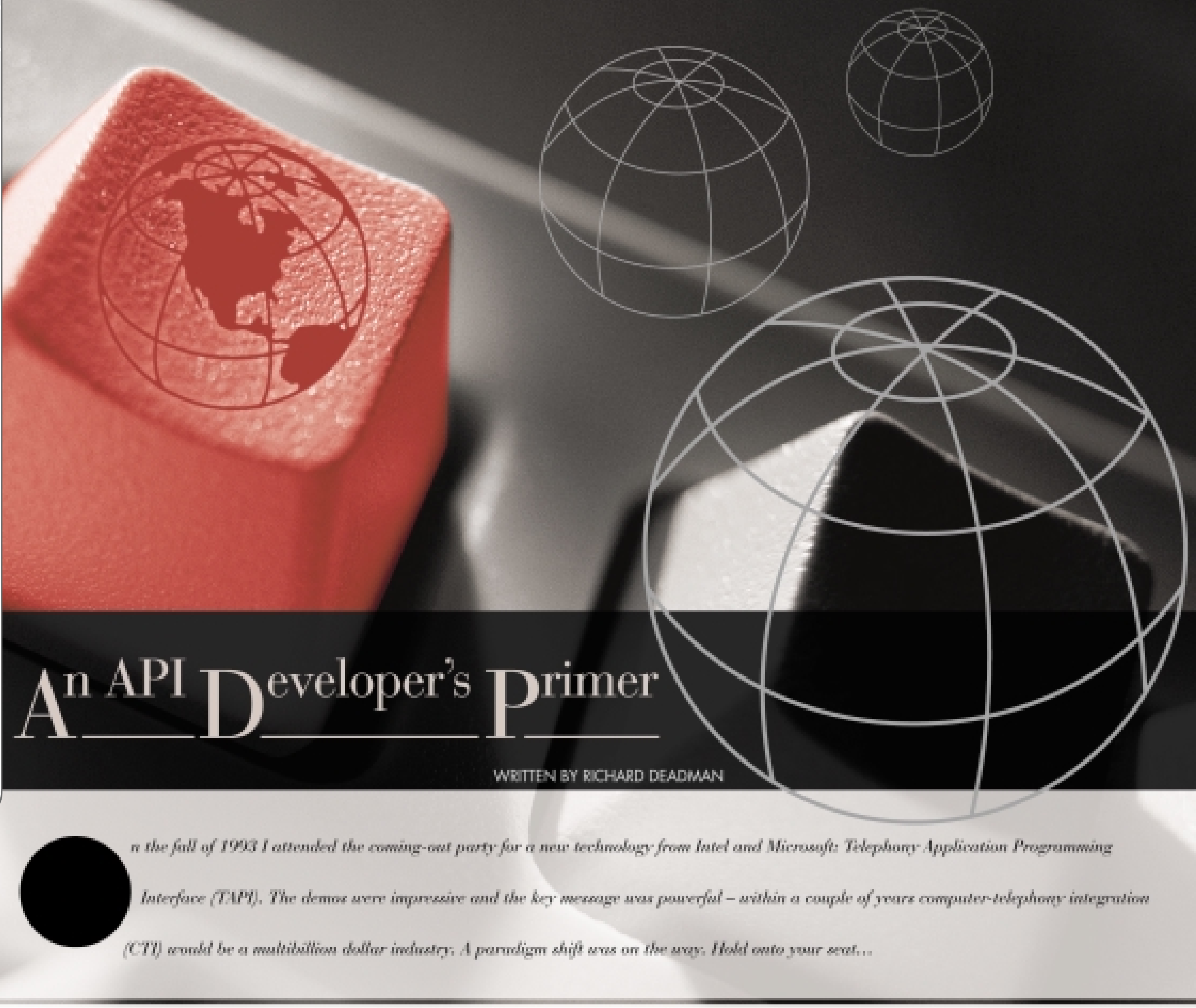
doctorjava@sys-con.com

appdev training company

www.appdev.com/promo/MG00052

installshield
software corp.

www.installshield.com



An API Developer's Primer

WRITTEN BY RICHARD DEADMAN

In the fall of 1993 I attended the coming-out party for a new technology from Intel and Microsoft: Telephony Application Programming Interface (TAPI). The demos were impressive and the key message was powerful – within a couple of years computer-telephony integration (CTI) would be a multibillion dollar industry. A paradigm shift was on the way. Hold onto your seat…

Of course, TAPI still has a niche and is indeed embedded into many PC modems, but its track record against its original goals is, shall we say, modest. Some would say that it never saw the Internet coming. Of course, technology development is almost Darwinian in the way it casts off also-rans. VRML promised us the world and is much cooler than HTML; Microsoft's Blackbird would give us our information superhighway; JSDT would revolutionize application collaboration; Network PCs, BeOS, BetaMax…

With the Java Community Process starting approximately 150 Java API definitions now, it shouldn't be surprising that not all of these will succeed. InfoBus 2.0, for instance, was withdrawn. Of course, success can be measured in different ways, and for some APIs even narrow adoption is appropriate. That said, some are bound to fail when judged against their original goals. Obviously JDBC and EJB are resounding successes, but what of JTAPI or JavaSpaces or Java Speech API? This article looks at what makes a good API, what pitfalls should be avoided, and who the winners and losers are.

Case Studies

Here's a sampling of Java APIs that have suffered from less than overwhelming adoption or are fraught with potential problems:

JSAPI

The Java Speech API targeted the same market as Microsoft Windows' SAPI. It basically offers capabilities for generating speech and performing speech-to-text. While the market has never been huge despite Corel's inclusion of speech recognition in its Office Suite, there seemed to be some potential for a JSAPI. Unfortunately, the number of implementations is somewhere around two. Why?

- ***Small market niche:*** Despite the hopes of speech recognition companies, this market is still small and specialized, with some inroads into automated directory assistance services.
- ***No published API classes:*** Indeed the API is published not as a set of interfaces and classes that implementors code against, but as a document that must be implemented. This makes implementation more difficult, leads to a high likelihood of typographical errors, makes it hard to develop to the specification without a working implementation, and makes it very hard to plug two vendor implementations into one system.
- ***Implementation isn't decoupled from specification:*** Part of the reason the API classes and interfaces are not published as code is that the specification embeds the implementa-

sitraka
software

www.sitraka.com/jclassSS/jdj

tion in its definition. That is, there's no concept of interfaces that a vendor writes to and plugs into an application using some factory method.

Preferences

At the time of writing, the Preferences API (JSR #10) has been accepted as part of J2SE 1.4. Its intent to provide a better registry-like replacement for properties is long overdue and especially topical as the era of distributed Java applications through mobile devices and Web services gains a new life.

That said, the Preferences API currently contains one fatal flaw: it ties the persistence of part of an application's state to a JVM. The Preferences class delegates its calls to a subclass of the AbstractPreferences class, which defines some Service Provider Interface (SPI) functions for completing the actual management of data between the application and the Preferences backing store.

Like AWT, the determination of which SPI will be instantiated behind the Preferences class is left up to the JVM implementation (a Java Property: java.util.prefs.PreferencesFactory is suggested, but not mandated). This means that an application has no way of plugging in an appropriate Preferences implementation – a networked implementation for an applet and a local registry for an application. Needless to say, it can't plug two implementations into the same application.

Most troubling, since the Preferences backing store is tied to the JVM implementation, you'll lose all your Preferences if you change JVMs. Sure, there's support for exporting and importing Preferences, but this would require users to manually migrate Preference information when they want to change their JVM vendor. Depending on your system "path," an application that can also run as an applet may not be able to share preferences between run types. It's one thing for the AWT to tightly tie the implementation to a JVM – GUIs are a transient aspect of an application after all – it's quite another to bind a persistence implementation to a JVM. On servers, in particular, it's not unusual to have several JVMs resident with different performance characteristics.

JTAPI

JTAPI started development in 1996 as a uniform way for applications to access telephony services, hard on the heels of TSAPI and TAPI in the "C" world. Since that time it has been through several revisions and, at the time of writing, the current release is 1.3. Unfortunately, even after all this time, the number of available JTAPI implementations can probably be counted on one hand. Some are only test frameworks that offer no real telephony access and, among the one or two released by reputable telephony companies, it's not unusual for them to "modify" the standard.

Why didn't it take off? Well, there are a number of reasons.

- ***Complex API***

JTAPI has to be one of the most complex APIs around, consisting of 19 packages with over 250 interfaces to implement. Implementing and maintaining all, or even part, of this is a major undertaking, which can only be justified by significant developer demand. This becomes a vicious circle: no applications mean no incentive to invest in JTAPI implementations, and with few implementations there's little incentive to build applications.

- ***Optional Hell***

JTAPI allows most packages to be optional, meaning a vendor doesn't have to implement them. On top of this, even within interfaces, many methods are also optional, permitted to raise the "MethodNotSupported" exception. While an obvious side-effect of trying to map a unified API to a complex and nonstandard industry feature set, this causes two problems. For JTAPI implementors, we must now try to predict our client requirements and trade that off against development complexity in order to determine which features to support. We must also implement a whole "Capabilities" framework to expose our feature set. For application developers, it provides extra uncertainty since it's not guaranteed that we'll be able to plug in multiple vendors to support our application.

The whole promise of JTAPI was decoupling applications from telephony providers, but in reality all it does is provide a common development paradigm for building Java telephony applications, since finding two JTAPI vendors with a set of intersecting features that match an application's requirements is rather hard (actually, finding two JTAPI implementations of any kind is no easy feat).

"Of course, technology development is almost Darwinian in the way it casts off also-rans"

- ***Shifting Sand***

Successful APIs offer gradual improvements. Unfortunately, JTAPI 1.3 saw listeners replace observers and a wholesale change in the way media was controlled. Even worse, while listeners were added to the core JTAPI package, subpackages such as "call control" didn't have listeners added yet. So, to receive "call control" and "core" call events, you either have to add both an observer and a listener to the call or just add an old style, deprecated observer.

- ***Broken Standard***

As was mentioned in the previous point, JTAPI changed the way media was controlled in 1.3. Unfortunately, the 1.3 API was rushed out before it was properly completed, as can be seen by the simple fact that the "media" API can't be implemented.

To control media, an application should instantiate a "MediaService" implementation, which by necessity must be tied into the JTAPI implementor's private code. But there's no "Factory" capability defined in the API, only a "BasicMediaService" class for subclassing. But BasicMediaService extends "NullMediaService" and so can't be tied easily to a real

loox
software inc.

www.loox.com

implementation. Real implementations are forced to replace the JTAPI API classes or provide proprietary solutions. Thankfully this is being fixed in JTAPI 1.4.

- ***Lack of CTI Market***

The Computer Telephony Integration (CTI) market, like many high-tech startups, is always "just about to explode." Maybe the arrival of the Internet has taken away the original vision of CTI. Nevertheless, while this is not a problem as we can attach to JTAPI directly, it does highlight how a complex API combined with a nonexistent market is a death blow.

- ***No Vendor Support***

JDBC offers a vendor-neutral view of relational data, and RDBMS vendors (and third parties) have endorsed it heartily. After all, what good is a database if applications can't access it? In addition, database vendors tend to market their products based on performance, price, and reliability.

The telephony market is very different. Telephony vendors market on feature sets ("Vendor X may support 'Park', but we support 'Park' and 'Camp-on'"), regardless of how few users know how to press the right combination of "*", "#", and numbers to use the features. A telephony system without application access is, well, still a telephony system. Considering the complexity of JTAPI, there's no compelling reason for most vendors to implement it.

- ***No Reference Implementation***

JTAPI has never provided a reference implementation against, for example, TAPI in win32. This leads to legions of frustrated developers flooding the JTAPI mailing list with questions about where they can get an implementation to run their code against. Eventually they go away and the potential vendors have no set of JTAPI applications to justify development of a JTAPI implementation. Chicken and egg. Even though the Java Community Process now requires reference implementations for Java Specification Requests, this requirement was unfortunately waived for JTAPI 1.4.

- ***Fragmented Market***

Finally, we should note that JTAPI is not even the only telephony API being developed under the Java Community Process. There's also a Parlay-based "service provider application" API and Jain Jcc and JCAT.

Despite this litany of problems, there may be some hope on the horizon. Recently there has been a push to rationalize the Jcc, Parlay, and JTAPI APIs so they're not isolated islands but build off each other. More important, JCC/JCAT is targeting the ongoing convergence of voice-into-network services and this is a space in which Java is attractive to implementers, much as Java has become firmly entrenched in the application server market. JCC/JCAT not only targets a real and growing market, it's also cut down significantly on the size of the API that vendors must implement. And JCC has been released with a reference implementation, so maybe the lessons are being learned.

Avoiding Pitfalls

These three case studies illustrate some of the problems that can doom an API to marginal use. It's important to understand these if we don't want to repeat past mistakes. Even in an internal development effort, a company will want to ensure that any reusable framework being developed for internal usage will be used, or else why put money into developing it?

1. ***Make sure the framework has a market:*** VRML is a cool technology, but no one could figure out how it could provide value to users.
2. ***Make sure the API has vendors:*** If you're developing an API to interact with telephony systems, the vendors of those systems are not going to write implementations to your API unless they want your users and are willing to conform to a standard. After all, in some markets, vendors sell based on their differences from other vendors, not their compatibility.
3. ***Entrenched opposition:*** Systems change slowly and you must show a significant benefit over existing systems in order to convince developers to risk moving to a new API. Databases are a case in point. Corporations have a large investment in relational databases and JDBC is mature, so any object-oriented API that connects to an object database is fighting an uphill battle
4. ***Complexity:*** There's always a tradeoff between power and simplicity. Try to enable developers and service provider implementors with a minimal set of core features.

"Often the source of failure has as much or more to do with shifts in the technology market"

Winners and Losers

Let's cut to the chase. Which APIs have failed to meet expectations and why? Note that Table 1 is a subjective list and I'm sure I'll take abuse for it. Furthermore, while some of these APIs have failed for technical reasons, often the source of failure has as much or more to do with shifts in the technology market.

This list is by no means exhaustive, but it does give you some idea of which APIs have taken the right steps. The success of Java on the server has also tilted the balance in favor of server-side APIs such as JSP, EJBs, and JNDI. Some APIs, such as Swing, generate such a vigorous debate that they can only be described as flawed successes.

The Case for Some Guidance

One of the most striking things about the Java APIs, once you've dug into the details, is the lack of consistency in their design decisions. Since each API has been developed by its own team, with only limited overall guidance, maybe this shouldn't be surprising. Some teams choose to support pluggable implementations using an explicit Factory pattern (e.g., isavvix corporation

java.isavvix.com

JTAPI), others hide the factory behind an obscure system property (e.g., AWT peers), while yet others use a URL lookup of registered providers (e.g., JDBC).

This means that developers must learn the usage paradigm of each API separately and usually can't take that usage pattern knowledge with them when they move to a new API. Now maybe this is justified – I do expect to have more control over what database driver to use than how the AWT renders my widgets – but nevertheless a little bit more consistency wouldn't hurt.

While the JCP does have a steering committee, it seems limited in its ability to approve APIs and resolve API overlaps. Java would benefit from a steering committee that provided an overall pattern framework for the APIs. For instance, they may define the preferred patterns for plugging vendor implementations into an API using a Factory pattern.

Under the Java Community Process 2, the process has been updated to correct some flaws in the original. In particular, JCP 2.0 requires that both a reference implementation and a compatibility toolkit be produced for the API. The requirement for a reference implementation has been successfully used before by the OMG in its CORBA standards and helps to spur on developers; this in turn makes vendors more interested in implementing the API. A reference implementation also ensures that an API is feasible. Similarly, a compatibility toolkit helps to keep API implementations from fragmenting. Unfortunately, the JCP 2.0 came along too late for JSAPI and JTAPI.

Even with these improvements, there are still some areas where the JCP could be improved. The JCP works on a consensual model, as opposed to the OMG adversarial model of Request for Comments and Request for Proposals. While this can speed up API development, in the real world it can also stifle options. Usually the specification lead comes from the group that first proposed the API, and it's not surprising that they may come into the process with some preconceived notions of an appropriate architecture.

Better and more consistent APIs might evolve if different groups could propose to the JCP Executive Committee (or better yet, a JCP Community Forum) alternative API architectures and patterns and have the API "direction" come under some competition. Once this high-level phase is resolved, the different sides could be brought together into a consensual group and proceed as they do now.

The Java Community Process has learned some valuable lessons since the JCP 1.0, but still it could benefit from better cohesion if:

- Platform design pattern standards were defined.
- More competition for initial API "architectures" was supported.
- A dependency map of technologies was

API	Reasons
DEAD	
InfoBus 2.0, JavaBeans Activation Framework	• While JavaBeans are a great way of providing a pattern for IDEs to use to inspect arbitrary widgets, the InfoBus idea of hooking these together through an asynchronous data bus never really caught on with developers or tool vendors. It's simply too easy to wire bean events to event handlers that manipulate other beans directly through code. Similarly, the JAF, while not technically flawed, envisioned a role for Java that has not materialized.
JSDT	• Market for distributed media Java applets and applications is small • Competes with RMI and CORBA as yet another distribution infrastructure
Open Service Gateway Specification	• No pressing need • Home networks are generally run by tech-savvy types who prefer the mature facilities provided by a Linux box
Orthogonal Persistence	• Object databases are a niche market • Competes with Java data objects
WOUNDED	
JSAPI	• Small market • No reference implementation • No interfaces to build against
JTAPI	• No reference implementation • Too complex • No incentive for vendors • Inconsistent and broken API
Preferences	• Couples an application's persistent state to a JVM implementation
JINI, JavaSpaces	• Yet another distribution framework • Cool idea but no driving need
VICTORS	
JSP/Servlets	• Targeted a need for server-side page generation with a clean language and server portability
EJB	• Popular on the server due to platform independence and vendor neutrality • Provides standard to unify server application framework that was already evolving to meet the need
XML Parser	• Hot technology • Pluggable architecture
JDBC	• Bridge to corporate data that drives many applications • Vendor-neutral • Vendors have incentive to provide drivers since they need applications to have access to their databases
Collections Framework	• While there was some controversy when the Collections Framework was being developed, especially for its refusal to reuse the popular JGL, it has succeeded due to its simplicity and, most important, its inclusion as a core Java package.
JNDI	• Ties into a popular industry niche of naming and directory services with a flexible and simple API. It doesn't hurt that the wildly popular EJB standard uses JNDI as its discovery mechanism.

TABLE 1 Winner and loser APIs

actuate
corporation

www.actuate.com/info/jbjad.asp

clearly laid out (I depend on EJBs, which themselves use JNDI, so I can use JNDI without inflicting extra dependencies on my applications).
- All JSRs required a reference implementation.
- Optional packages and methods made up a minority of the API.
- APIs were accompanied by usable classes the developers can import into their development environment and code against.

Of course, this whole discussion also applies to the creation of APIs by other parties, internal to an organization or not. Using RFPs in the development of a proprietary API might not make as much sense, however. And getting open source projects organized enough to adopt RFPs on a wide scale may be a bit optimistic.

Lessons

Despite my criticisms, I should point out that the standard Java APIs, like many proprietary APIs and APIs in other languages, are the result of a lot of hard work by bright people. When an API is flawed, as opposed to just not adopted widely, it's often due to the lack of an overall guiding framework that ensures that the API works well with other APIs and fits into a coherent usage paradigm. For a successful API, here are some simple rules:

1. Find an industry niche with a high value for developers, high enough to offset the extra cost of learning a new API. An example is JDBC. Of course, as the Java "platform" matures, these niches tend to get narrower and narrower until a new technology bursts on the scene (e.g., XML, SOAP).
2. Hide complexity from the user of the API, but not so much complexity that the API is hard to implement. The harder it is for vendors to implement, the fewer implementations there will be. This prevents developers from using it, which in turn discourages vendors from implementing it. Both JTAPI and JSAPI (speech) have suffered from a dearth of implementations, partly due to the complexity of the API implementation. On the other hand, HTML and HTTP are examples of standards that are wildly successful because they were just complex enough to be useful.
3. Ensure that the vendors are there. While telephony vendors were actively involved in developing JTAPI, there wasn't a lot of incentive for them to actually build implementation. Traditional PBXes and Switches have been marketed on their differentiating features instead of performance, quality, or price, which doesn't lead telephony companies to want to present an API that hides this differentiation.
4. When choosing between sticking to a widely adopted pattern or implementing a "better" but unusual pattern, stick to the pack. There are various pluggability patterns employed by APIs to hook applications to API implementations, such as a Factory pattern or registry, broker, or system property. Unfortunately, there are too many of them.
5. When an API manages the persistent aspects of an application, ensure that it's not tightly tied to the JVM. It's one thing for AWT to be tightly bound to the JVM, quite another for the Preferences API to be bound to the JVM.

Parting Shot

While I've concentrated on the well-known Java Community Process APIs (familiar to most developers), the same lessons apply to internal proprietary APIs as well as non-Java APIs. The pitfalls and lessons discussed here apply just as much to internal development efforts as to public ones. After all, even inside a company a successful API still has customers who must be kept happy.

The current rush to XML standards – which are not programming APIs but data formats – provides an eerie parallel to this discussion. In fact, in reviewing the debris being left on the Java API highway, it becomes obvious that while there are technical errors, the most common reason an API dies or doesn't reach its anticipated potential is that the technology market shifts, leaving InfoBus and JAF wondering what happened to visual JavaBean assemblers even as the market moved toward server-side Java.

There's nothing about Java, compared to other languages or platforms, that has led to poorer APIs, and a lot has improved. Object-oriented code enables easy pluggability, interfaces provide nice decoupling between the API and the implementation, and packages are extremely useful namespace separators. If I appear critical of Java APIs, it's like the criticism you may give to a beloved child. Since mistakes are always easier for armchair critics to find afterwards, it should be noted again that most Java APIs are well crafted and that the efforts of the API designers is and has been diligent and thoughtful.

But there's always room for improvement.

Resources

- *JSAPI:* http://java.sun.com/products/java-media/speech/
- *Preferences API:* http://java.sun.com/j2se/1.4/docs/guide/lang/preferences.html and www.jcp.org/jsr/detail/10.jsp
- *JTAPI:* http://java.sun.com/products/jtapi/index.html
- *InfoBus:* http://java.sun.com/products/javabeans/infobus/
- *JavaBeans Activation Framework:* http://java.sun.com/products/javabeans/glasgow/jaf.html
- *JSDT:* http://java.sun.com/products/java-media/jsdt/index.html
- *JINI:* www.jini.org/
- *JavaSpaces:* http://java.sun.com/products/javaspaces/
- *Open Service Gateway Specification:* www.jcp.org/jsr/detail/8.jsp
- *Orthogonal Persistence:* www.jcp.org/jsr/detail/20.jsp
- *JSP:* http://java.sun.com/products/jsp/
- *Servlets:* http://java.sun.com/products/servlet/index.html
- *EJB:* http://java.sun.com/products/ejb/index.html
- *XML:* http://java.sun.com/xml and http://java.sun.com/xml/jaxp/index.html

Author Bio
Richard Deadman is a consultant in distributed Java systems with interests in metainformation and dynamic business logic. He holds an MS in computer science.

rdeadman@deadman.ca

intraware

www.intraware.com/portal

Extending the J2SE 1.4 Logging Package

Monitoring made easy

WRITTEN BY
Jim Mangione

Many applications require, from initial development through postproduction, a reliable means of monitoring the state of the software and its data. Users, support staff, and engineers can capture and analyze points where state transitions occur such as changes in persistent or nonpersistent data elements, or custom-defined areas of interest.

The most basic way to capture these elements of interest is through application logs. Most Java-based production systems have them in some form, and most of them probably implement a custom API or use one of a handful of third-party packages that may or may not be cross-compatible. Out comes java.util.logging in the new Java 2 Platform, Standard Edition (J2SE) v1.4. Developed collaboratively with input from several key contributors (see "JSRs: Java Specification Requests" at http://jcp.org/jsr/detail/47.jsp for details), this package can be used as is, extended for additional functionality, and in conjunction with enterprise application services.

How does it work out of the box? What are its limitations and how easy is it to extend its capabilities? I'll discuss these issues, plus show how to add database-level logging to the package's framework.

J2SE Logging: Out of the Box

Let's start by discussing what can be done with the basic logging package. I'll give a general overview, but I suggest you take a trip to http://java.sun.com/ for a more complete picture.

Here's how it works: the package provides an API for producing a LogRecord. This record is what the logs are populated with; it contains such properties as the datetime, log level (there are seven currently, from FINEST all the way to SEVERE), and, of course, the message itself.

Formatting and output for log records are done through Formatter and Handler classes. Formatters determine which format the LogRecord will be in, currently either plaintext or XML (the default). Handlers define how the logs are exported (through a file, a socket, to the console, or in-memory). To set up a Handler class to point to a file called "mylog.xml" that writes each LogRecord as an XML message, you'd use the FileHandler class and set its formatter to an instance of an XMLFormatter, as follows:

```
FileHandler flhandler        = new
  FileHandler( "mylog.xml" );
XMLFormatter xmlfrmtr        = new
  XMLFormatter();
flhandler.setFormatter( xmlfrmtr );
```

To integrate logging into an application you must obtain a static instance of the Logger class. Through this class a user will define the output type, specify a log format, and publish the individual log records.

To publish an informational level message in the "mylog.xml" file, you first need to retrieve a Logger for your subsystem (commonly the class itself), then attach the handler to it (in our case it's the FileHandler). Now you're ready to generate messages.

```
private static Logger logger =
  Logger.getLogger("com.mylogger.
  TestDriver");
logger.addHandler( flhandler );
logger.info("TestDriver(): Testing
  INFO");
```

For the logger.info() method, you can also use the more generic logger.log(), which takes Level as a param, as well as the message:

```
logger.log(Level.INFO, "TestDriver():
  Testing INFO").
```

The code above will produce an XML <record> message with each field in the LogRecord shown as a separate tag:

```
<record>
  <date>2001-11-
    20T20:21:03</date>
  <millis>1006305663460</millis>
  <sequence>0</sequence>
  <logger>com.mylogger.TestDriver</log
    ger>
  <level>INFO</level>
  <class>com.mylogger.TestDriverFile
    </class>
  <method>main</method>
  <thread>10</thread>
  <message>TestDriver(): Testing
   INFO</message>
</record>
```

A complete example of how logging is added to an application appears in Listing 1. (Listings 1–5 can be downloaded from www.sys-con.com/java/sourcec.cfm.)

What isn't obvious in this example is the presence of a LogManager. This class contains default configurations for loggers and their supporting classes (read in from a logging.properties file), and is a single global entity shared by each logger within the application. It also maintains a list of global handlers to which, by default, each logger sends its output.

Extending Its Capabilities

As with any thorough framework, the logging package can be extended to increase its capabilities and provide custom functionality on how to format and generate logs. Perhaps e-mailing levels of quintessence systems limited

www.in2j.com

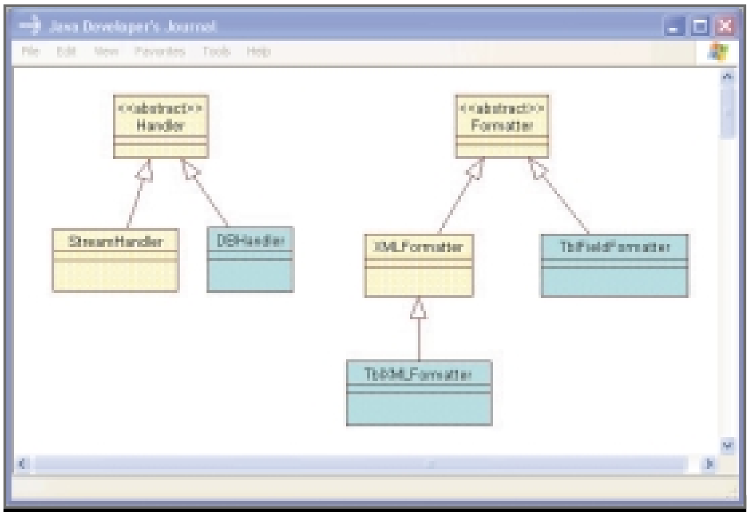


FIGURE 1 Diagram of handler and formatter

SEVERE to administrators or providing the ability for other applications to query and report on the logs is necessary. In the latter case we can replace the common log file with a simple database table. I'll demonstrate how to do this by extending both the Handler and Formatter classes, giving total control of where the database is located and how to write a log record to it.

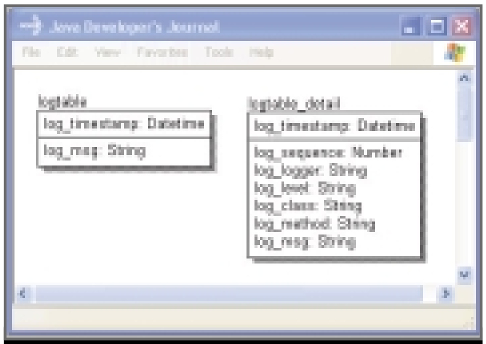
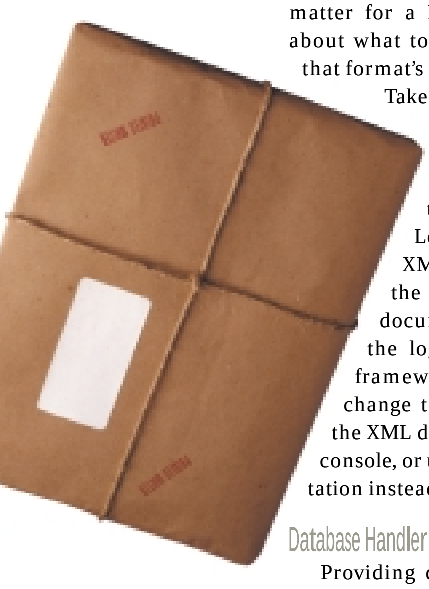


FIGURE 2 Model of two database tables

First, it's important to understand the class diagram, which shows what we'll be extending. Referring to Figure 1, you'll notice an abstract formatter and handler. The formatter decides only how the LogRecord will look. It works on the individual properties of the LogRecord and creates a single string (usually) to be returned that reflects a complete record in a particular format. The handler, which calls the formatter for a LogRecord, worries only about what to do with the outcome of that format's record when returned.



Take the common FileHandler, for instance. This class specifies the filename to which logs are written, then for each incoming LogRecord calls the default XMLFormatter to retrieve the entire record as an XML document and write it out to the log. With the ease of this framework, it's only a small change to switch from outputting the XML document from a file to the console, or using a plaintext representation instead of XML.

Database Handler and Customized Formatters

Providing database support to this logging framework will involve implementing only two classes: a formatter to build the SQL necessary to insert an entire LogRecord into the database, and a handler to do the transactions. We'll also include a third class, another formatter that will split each field in the log record into separate database fields.

As shown in Figure 1, the three new classes are DBHandler, TblXMLFormatter, and TblFieldFormatter. How do they work together? The user instantiates a DBHandler and passes the JDBC driver being used and the appropriate connect string. In this case we're using MySQL:

```
DBHandler dbhandler = new
DBHandler("org.gjt.mm.mysql.Driver",
  "jdbc:mysql://localhost/logger?
   user=jim&password=changeme");
```

Then, instead of using the XmlFormatter, a new TblXmlFormatter is instantiated and attached to the DBHandler. This will return column names and LogRecord values in a format suitable for inclusion in a database transaction.

```
TblXMLFormatter tblxmlfrmtr = new
  TblXMLFormatter();
dbhandler.setFormatter( tblxmlfrmtr );
```

The handler will also write a default table name, which can be changed. The schema of this table depends on which formatter is used. If the user wants the log to consist of XML documents, the TblXmlFormatter is used; this will populate a single column with the entire log (see Figure 2, Logtable). If individual columns have to be populated, the TblFieldFormatter will do the job (see Figure 2, logtable_detail). This class inserts each field of the LogRecord into a separate column and can also control whether a field is written to the table.

Both formatters allow for changing the column names of the table, just as the handler allows for changing the table name. This provides flexibility in the location the logs are written to and allows integration into an existing enterprise schema, if required.

DBHandler

Implementing a new target to send logs to requires developing a new Handler class. For the target of a database table, this class should encapsulate all necessary logic for connecting to the database, inserting formatted log records into the specified table, and handling any SQLExceptions.

For the DBHandler we'll inherit directly from the Handler base class. This requires minimally implementing the publish(), close(), and flush() methods, as well as a constructor that will set up our connection.

The first step will be designing that constructor. Since we want this class to be database-independent, we'll need to pass in the JDBC driver that will be used, plus the connect string, including username and password. Some standard configuration activities are required next, such as setting the LogManager and default Level and Formatting. Then the database-specific code is executed, which involves setting class-level variables to the driver and URL, and establishing and holding a connection in another variable.

```
public DBHandler( String driver,
  String url )
    throws SQLException,
      ClassNotFoundException,
      SecurityException {
    // … other configuration code
      here
    this.jdbcdriver = driver;
    this.jdbcurl    = url;
    dbconnect(); // connect to the
      database
  }
private void dbconnect()
    throws SQLException,
      ClassNotFoundException {
```

preemptive solutions

www.preemptive.com

```
    Class.forName(jdbcdriver);
    this.conn = DriverManager.
     getConnection(jdbcurl);
  }
```

This connection will be used for each transaction performed by the publish() method, where the bulk of the work is done. This method is responsible for dynamically creating an insert statement from the formatter, executing the transaction, and catching any SQLExceptions.

To do this, it expects the formatter to return both the table column names and the log values. The column names are a static string obtained by the header portion of the formatter (to be discussed in the next section). The log values are obtained by passing in the LogRecord and getting back a single string containing each column's value. The logic looks like this:

```
public void publish( LogRecord record
     ) {
   // …
   Statement stmt = conn.
    createStatement();
   String sql = "insert into " +
    tblname + " (" + frmtr.getHead()
     + ") values (" +
     frmtr.format(logrecord) + ")";
   stmt.executeUpdate( sql );
   stmt.close();
   // …
}
```

Why not pass the entire insert statement back from the format() method? That could be done, but it requires the formatter to know the name of the table. This would break the rules of encapsulation for that class, since the formatter should only work with a LogRecord and not have to worry about the table name or database information.

Another design decision was to stay away from prepared statements. Using them would require an added level of complexity since we'd need to return individual values to set them rather than a single string, which is what format() is designed to do.

The flush() method isn't applicable to this type of handler, so no implementation is present. For any of the StreamHandlers, however, this would flush the writer. The last required method, close(), will attempt to close the database connection.

A complete listing of this class can be found in Listing 2.

TblXMLFormatter

This formatter will need to produce two columns to incorporate into the SQL generated by the DBHandler. The first column will be the timestamp, which will use the standard System datetime. The second column will be a large string containing the entire XML message.

To extend the formatter requires implementing format() and, optionally, getHeader() and getTail(), which return an empty string by default. Although we can use the base class formatter to create our new class, you'll notice we're inheriting directly from the XMLFormatter class, so we can take advantage of the code that already produces the XML from the LogRecord.

First, we have the getHeader() method, which is responsible for returning the names and the order of the columns being returned.

```
String returnColNames = col_time
  stamp+","+col_msg;
```

If we use the default column names from this class, the SQL in the DBHandler will go from this:

```
"insert into LOGTABLE (" +
  frmtr.getHead() + ") values (" +
  frmtr.format(logrecord) + ")";
```

to this:

```
"insert into LOGTABLE (LOG_TIMESTAMP,
  LOG_MSG) values (" +
  frmtr.format(logrecord) + ")"
```

Next, in the format() method, we call super.format() to extract the XML and concatenate it with the system datetime that will be returned. This matches the order of what getHeader() returns. Also note that we're using the JDBC escape sequence for the timestamp to ensure database independency.

```
java.sql.Timestamp tm = new
 java.sql.Timestamp(System.currentTime
 Millis());
// include timestamp, plus xml
  message from super
String returnVals = "{ts '" +
  tm.toString() + "'},'"+
  super.format(record) + "'";
```

After format() is called, the final SQL in DBHandler will look similar to this:

```
"insert into LOGTABLE (LOG_TIMESTAMP,
  LOG_MSG) values ( {ts '2001-11-25
  18:25:00.61'}, '<record>xml message
  elements</record>')"
```

The complete code for this class can be found in Listing 3.

TblFieldFormatter

This formatter is provided to give the user the option of using only a selected number of fields from the LogRecord and separating them into individual columns in the database.

As with the TblXMLFormatter, this class also overrides getHeader() and doesn't use getTail(). Since the behavior of format() is entirely different from SimpleFormatter or XMLFormatter, we extend the Formatter class directly.

For getHeader(), we again return the names and order of the columns being used. Extra logic must also be introduced to provide the flexibility of turning on/off columns, but the order must remain consistent.

The format() method must also apply logic to determine which columns to display, and also to keep the order consistent with what is returned by getHeader(). Unlike TblXMLFormatter, we must manually extract each field from the LogRecord. Following is an example of how to include Level directly from the LogRecord, but only if enabled.

```
if ( enablecol_level )    returnVals
 += ",'" + record.getLevel
 ().toString() + "'";
```

Finally, as with TblXMLFormatter, there are get/set methods that allow changes to the names of the database columns, and an additional "enable" method per column for enabling/disabling their output.

Listing 4 provides the entire code for this class; Listing 5 shows how this formatter is incorporated into our original example application.

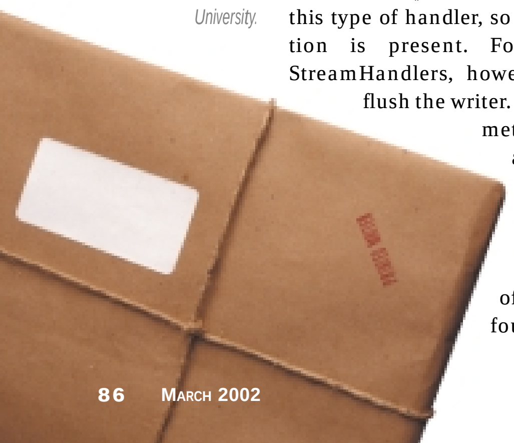
Conclusion

By adding the foregoing database capabilities to the new J2SE logging package, we've expanded the target of where logs can be stored and increased our options for how to query them. Your logs can now fit into any JDBC-compliant enterprise database schema and use standard database tools to produce reports.

It was my intention in this article to give insight into this framework and outline the steps necessary to customize where your logs go and how they look when they get there. This framework is a powerful utility in the new J2SE, whether used out of the box or customized to your specific requirements. I'll certainly be watching how it matures over time!

jmangione@carescience.com

Author Bio
Jim Mangione is director of software engineering for the Report & Query Division of CareScience, located in Philadelphia. Jim has over 12 years' experience in client/server and Web-based application development and holds a master's degree in computer science from Drexel University.



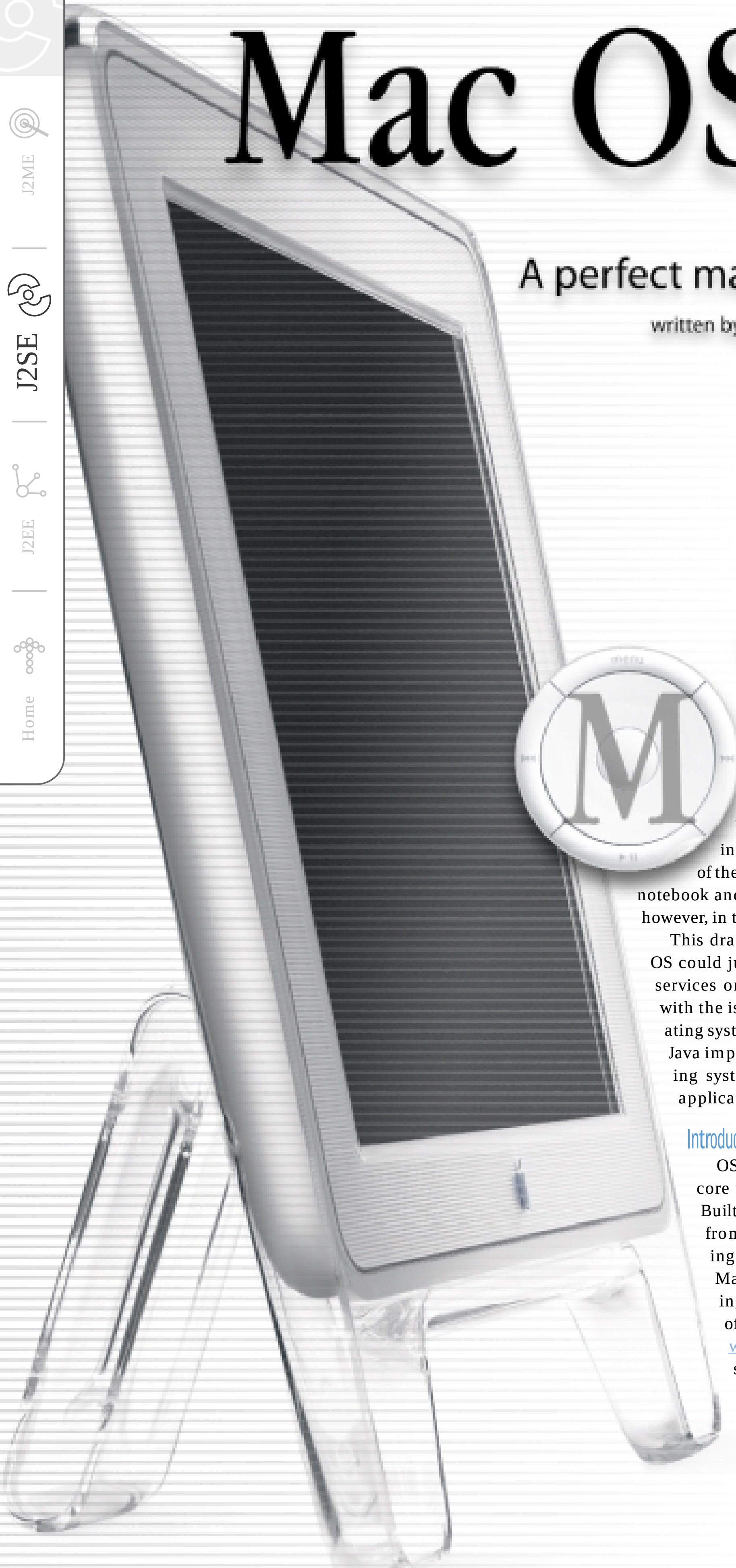
new atlanta communications

www.newatlanta.com

Mac OS X & Java

A perfect marriage

written by Hitesh Seth



Mac OS X sports a new look, not just on the outside with its great look and feel but also on the inside. OS X is proudly built on top of a BSD Unix-based core foundation. An exciting aspect of the new operating system is that the latest version of the Java 2 platform (J2SE v1.3) is preinstalled in every Macintosh notebook and desktop preloaded with OS X. (The current release is 10.1, however, in this article I'm referring to the latest version of OS X, v10.1.1.)

This dramatic shift in Apple's strategy toward Java support in Mac OS could jump-start the deployment of Java-based applications and services on a user-friendly consumer operating system, specifically with the issues surrounding Java support in the new Microsoft operating system, Windows XP. This article reviews the features of Apple's Java implementation and explores what makes OS X a great operating system for the development and deployment of Java-based applications.

Introducing Mac OS X

OS X is what a lot of developers want – a familiar Unix-based core with a highly productive and great-looking user interface. Built on top of BSD Unix, OS X represents a significant change from its predecessor, Mac OS 9. Darwin, the core of the operating system, combines the services provided by Unix and the Mac 3.0 kernel with support for high-performance networking capabilities. Apple took a big step toward the adoption of Darwin by making it an open source project (www.darwin.org/). Darwin supports a flexible model that in turn supports multiple file systems, including a Universal File System, ISO 9660 (for CD-RW disks), Universal Disk Format (for DVD volumes), and Mac OS Standard HFS. Key to Darwin is support for standards-based connectivity including TCP/IP, PPP, HTTP, FTP, DNS, DHCP, LDAP, and NTP.

Layered on top of Darwin are three graphics subsystems:

hit software

www.hitsw.com/xmlrdb

1. ***Quartz:*** A lightweight window server and PDF-based 2D graphics rendering library
2. ***Open GL:*** A 3D rendering model
3. ***QuickTime:*** For multimedia capabilities

To provide compatibility with its predecessor, Mac OS 9, OS X also supports a "classic" mode, which makes the system available in a dual operating system mode by running both the OS 9 Classic and OS X operating system. (It's actually possible to configure OS X to boot with the two operating systems at start up to avoid the delay of a later start up.)

From a development perspective, OS X provides multiple options – Carbon, the traditional Mac OS API that supports completely backwards compatibility with OS 9, and Cocoa, Apple's next-generation development framework for OS X "native" applications. A key component to OS X and the focus of this article is Apple's support for the latest version of the Java platform. OS X 10.1 supports Java 2 Standard Edition (J2SE) version 1.3.1, and HotSpot, the latest production version available for Java deployment. Java is treated as a first-class citizen in OS X and, along with Cocoa, is expected to represent a key framework that developers will mostly adopt to build applications for OS X.

The beauty of OS X is its most visible piece, Aqua, OS X's new user interface. Even though Mac OS is traditionally known for its ease-of-use and intuitive GUI, Aqua makes OS X stand out and represents a dramatic user-interface paradigm shift for an operating system. (Figure 1 shows the various components of the OS X architecture.) A key highlight of Aqua is the intuitive dock (as shown in Figures 2–4 toward the bottom of the screen). Having worked in multiple user-interface environments before, I found Aqua has a huge leg up with respect to usability and user friendliness.

OS X and Java

As we've seen in the previous section, support for Java in OS X is strategic to Apple's new operating system. Every installation of OS X includes a preinstalled version of J2SE 1.3 VM runtime and command-line tools. This is important as developers can now back the latest production-quality Java technologies in the core platform. It's quite refreshing to start a terminal session in OS X and type "Java" or "Javac" and find out that J2SE is preinstalled and configured on the OS. Table 1 summarizes some key environment characteristics of OS X support for Java.

A distinguishing factor is that OS X doesn't just embed Java into the core operating system, it actually takes the integration to the next level. For instance, a key highlight of OS X's Java support is the native preemptive multitasking threads support that can have a true impact on the performance of Java applications and allows developers to utilize symmetric multiprocessing-based systems.

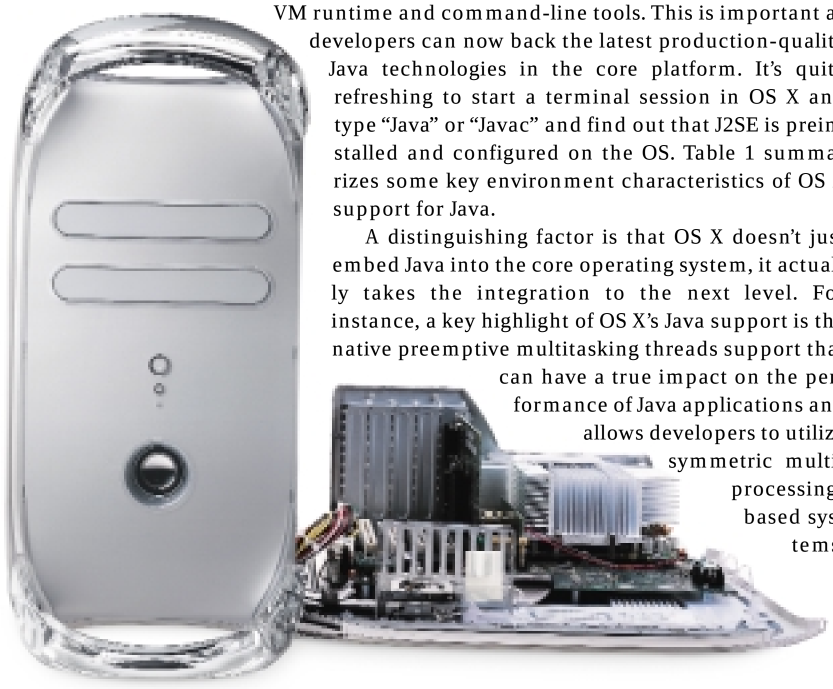
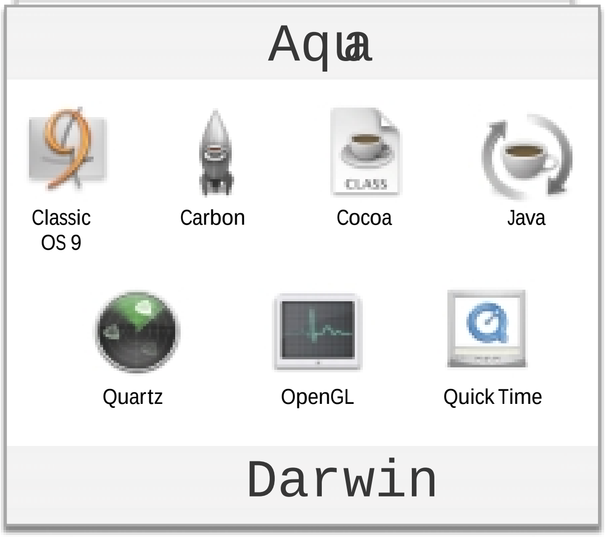


Table 1 OS X support for Java – key environment characteristics

Characteristic	Value for OS X (obtained from java.lang.System.getProperties())	
Operating System	os.name	Mac OS X
	os.version	10.1
J2SE Version	java.vendor	Apple Computer, Inc.
	Java.version	1.3.1
	JRE Version	Java HotSpot Client VM build 1.3.1
JAVA_HOME Directory (java.home)	/System/Library/Frameworks/JavaVM.Framework/Home The directory is also symbolically linked as /Library/Java/Home	

Figure 1 Mac OS X architecture



Another great runtime extension to Java in OS X is the ability to share class files and HotSpot compiled code across multiple invocations of the virtual machine. Implications of this feature can be significant, as it can be a boon to interactive Swing-based applications, which are typically memory intensive. Shared code can remove a lot of the overhead associated with reloading the large number of classes required for production-quality Java applications.

Key Highlights of Java Support in OS X

- ***J2SE 1.3:*** Includes the HotSpot Java Runtime Environment (JRE), Java compiler, and other command-line tools.
- ***OS X Look and Feel:*** An Aquafied implementation of Swing on OS X that gives Swing applications the brand new OS X look and feel.
- ***Java Web Start:*** An exciting new technology that holds a lot of promise (see Sidebar). To use Java Web Start on OS X, you can use applications developed by third-party vendors who use this technology. For instance, UML-based modeling has gained wide acceptance in the developer community. ArgoUML, an open source UML-based modeling environment, is quite useful and is packaged as a Java Web Start application. Figure 2 shows ArgoUML in action with Java Web Start.
- ***QuickTime for Java:*** A set of cross-platform APIs that allows Java developers to build multimedia, including streaming audio and video built on top of QuickTime 5, into Java applications and applets.
- ***Java Spelling Framework for OS X:*** A set of JavaBeans that adds spell-checking to Swing applications.
- ***Java Speech Framework for OS X:*** A set of JavaBeans that adds speech synthesis and speech recognition to Java applications.
- ***Cocoa Application Framework:*** An evolution from NeXTStep APIs, Cocoa is a collection of advanced, object-

pointbase, inc.

www.jdj5.pointbase.com

oriented APIs for the development of Aqua-based applications for OS X using Objective C and Java. Apple includes a set of Java classes that allows Java developers to build multimedia-enriched applications based on the OS X-specific Cocoa framework.

- ***Interface Builder:*** Another tool that's part of the Developer Tools CD, it can visually create the user interfaces used by Cocoa applications, then the associated event can be linked using Java code. The rule of thumb is to use Swing-based components for cross-platform compatibility, and Cocoa to utilize Aqua-based interfaces and target OS X-specific deployments.
- ***Project Builder:*** Supports Java development in Apple's Project Builder IDE.
- ***JNI:*** Supports invoking native OS-specific API and embedding Java into native applications using standard Java Native Interface (JNI) programmer interfaces.
- ***JDBC:*** Pure Java (or Type 4) JDBC drivers can be used to access database systems such as Oracle.

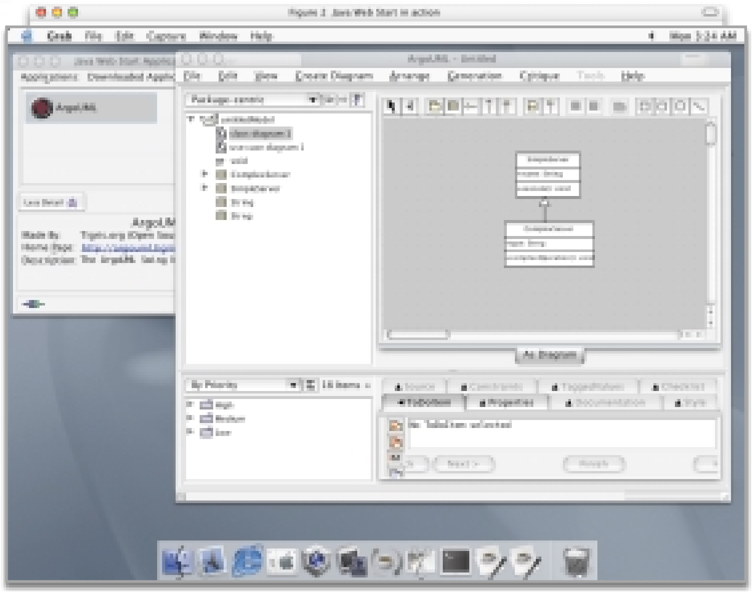


Figure 2: Java Web Start in action

Developing Java Applications Using OS X

OS X is very Unix-friendly as a development environment. All the utilities that we're familiar with on other Unix platforms (typically Sun Solaris, HP-UX, and Linux-based servers) are available in OS X. This benefits developers who intend to use traditional Sun Solaris, HP-UX, and AIX environments for high-end Java deployment but would like to have the ease-of-use of OS X. For a lot of developers (including me) who still like to use command-line tools, there's good news: vi and emacs, along with other Unix-style utilities such as awk, sed, and shell scripting (OS X includes multiple variants of shell implementation including sh, csh, tcsh and zsh), are all included in OS X. For developers who like the power of X-Windows, a port of X-Windows System is also available for OS X.

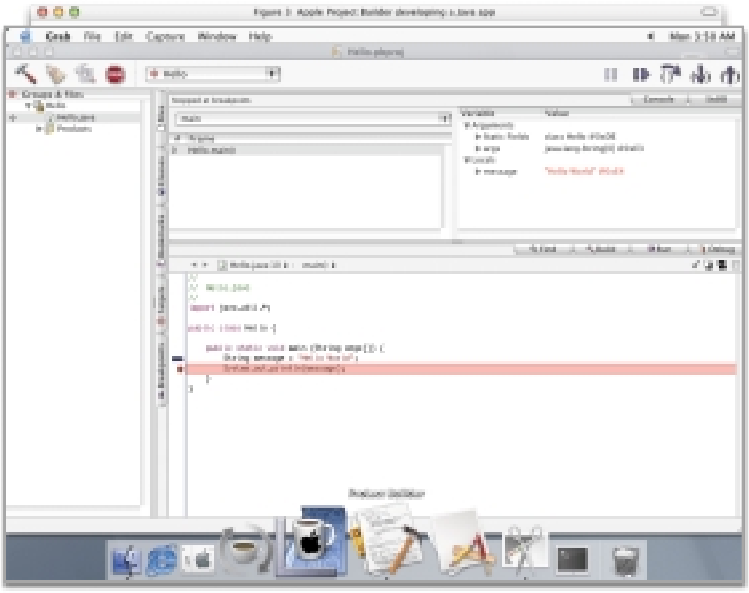
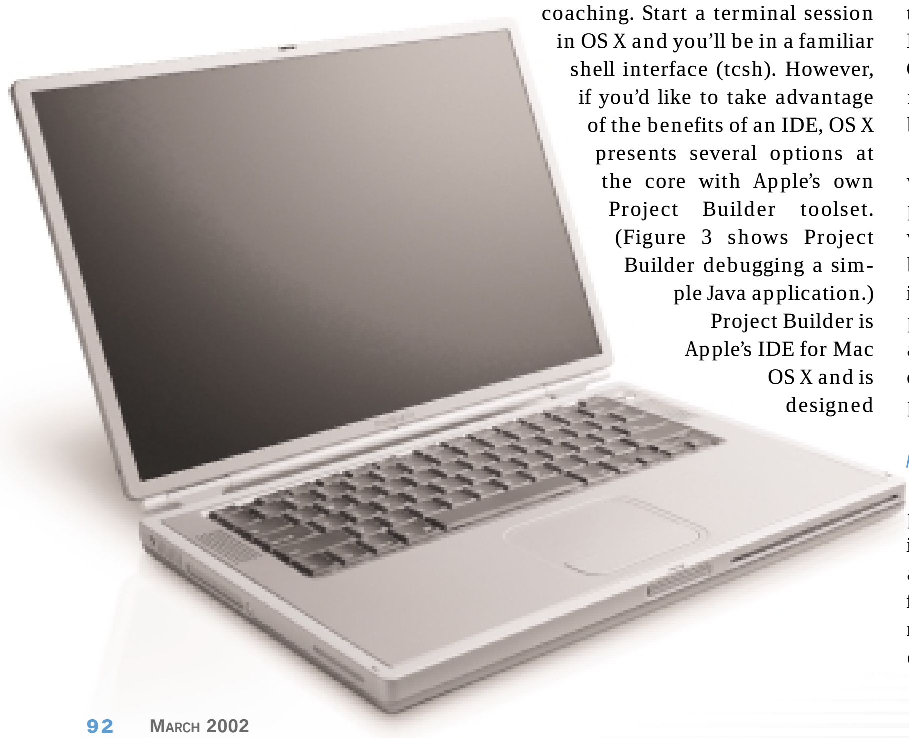


Figure 3: Apple Project Builder developing a Java app

If you work with tools such as vi/emacs to develop Java code, you don't need any further coaching. Start a terminal session in OS X and you'll be in a familiar shell interface (tcsh). However, if you'd like to take advantage of the benefits of an IDE, OS X presents several options at the core with Apple's own Project Builder toolset. (Figure 3 shows Project Builder debugging a simple Java application.) Project Builder is Apple's IDE for Mac OS X and is designed to support multiple development frameworks and languages. It supports C, Objective C, C++, and Java development using Carbon and Cocoa frameworks and Java, and also includes a new set of application-packaging mechanisms for easy distribution.



Project Builder is available on the Developer Tools CD, which is included with every shipment of OS X. From an IDE perspective, Project Builder supports core features, including workspace-based project editing, source-code management based on CVS, search and navigation, file editing, and building, running, and debugging facilities. From a Java developer's perspective, Project Builder supports the Java class browser and the WebObjects platform, and has the integrated ability to create clickable applications for the OS X platform. Its great performance and ease of use are instrumental in its usability.

Apple Project Builder

Apart from the core facilities of an IDE, Project Builder is pretty basic. For users interested in using an integrated user interface design, integrating with UML modeling tools, and wizard-based application generation capabilities, OS X has support from a host of third-party vendors and organizations. Borland recently released JBuilder 6.0 for Mac OS X for developing and deploying Java applications, JavaBeans, applets, JSP, and servlets.

visual mining

www.visualmining.com/jdj

It features an integrated set of tools including an editor, debugger, compiler, and visual designer. Apart from this, Code Warrior from Metrowerks is a popular IDE on a Mac environment. NetBeans, the Java-based open source IDE from Sun Microsystems, has been successfully tested on the OS X platform (www.netbeans.org provides the instructions to run NetBeans on OS X). Running Forte for Java 3.0, a commercial IDE based on NetBeans 3.2 technology, is very similar. Since OS X completely supports J2SE 1.3.1, you can potentially use any other IDE built using Java as well.

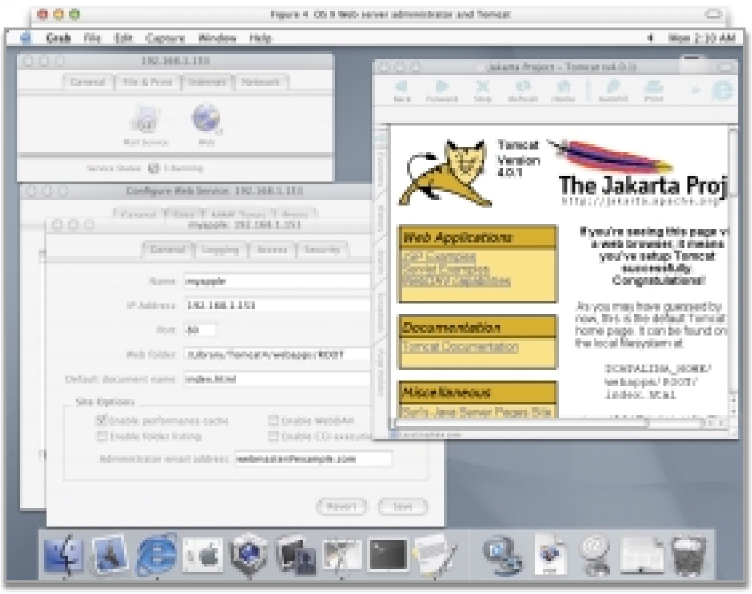


Figure 4 OS X Web server administration and Tomcat

Name: myapple
IP Address: 192.168.1.151
Port: 80
Web folder: /Library/Tomcat/webapps/ROOT
Default document name: index.html
☒ Enable performance cache
☐ Enable folder listing
☐ Enable WebDAV
☐ Enable CGI execution
Administrator email address: webmaster@example.com

If you're seeing this page via a web browser, it means you've setup Tomcat successfully. Congratulations!

As you may have guessed by now, this is the default Tomcat home page. It can be found on the local filesystem at:

$TOMCAT_HOME/webapps/ROOT/index.html

Grab File Edit Capture Window Help
General File & Print Internet Network
Web
Configure Web Service: 192.168.1.111
General Logging Access Security
Site Options
Revert Save
Jakarta Project - Tomcat 4.0.1
Back Forward Stop Refresh Home AutoFill Print
Tomcat Version 4.0.1
The Jakarta Project
http://jakarta.apache.org
Web Applications
JSP Examples
Servlet Examples
WebDAV capabilities
Documentation
Tomcat Documentation
Miscellaneous
Sun's Java Server Pages Site

Server-Side Java and OS X

As the name suggests, OS X Server is the server version of the OS X operating system. Key highlights of OS X Server include support for file-sharing services using Apple AFP, Windows (using Samba), FTP, NFS, Mail (IMAP, SMTP, POP3), and print services; Web services using Apache Web Server with WebDAV, PHP, CGI, WebObjects, and SSL support; QuickTime-based streaming media services; and open standards–based TCP/IP services, including DNS, DHCP, SLP, LDAP, graphical installation, configuration, and administration.

Mac OS X Server (version 10.1) extends the philosophy of using and embedding Java technologies. It automatically installs a preconfigured version of Apache Web Server (version 1.3.20) and Apache Jakarta Tomcat (version 3.2.3), the most popular reference implementation of server-side Java technologies (servlets and JSP in the /Library/Tomcat folder). To configure the OS X Web Server to work with Tomcat, uncomment the three lines in the Apache configuration file (etc/httpd/httpd.conf) to load the mod_jserv module, run Tomcat (/Library/Tomcat/bin/tomcat.sh), and restart the Web server using the server administration tool. Optionally, the Web server root can be set to Tomcat root application directory (/Library/Tomcat/webapps/ROOT/) to test the Tomcat instance. Figure 4 shows Tomcat in action on OS X and the Web server administration interface.

Keep in mind that the bundled Tomcat installation is version 3.2.3. If you need support for the new Servlets 2.3 and JSP 1.2 API (part of J2EE 1.3), download the latest version of Tomcat (v4.0.1) from http://jakarta.apache.org/tomcat. It involves replacing the 3.2.3 folder with the latest version and updating the httpd.conf configuration file with the new mod_webapp module provided by the Tomcat 4.0.1 binary distribution.

Tomcat can not only be used to serve dynamic Java applications from OS X, but can be a very effective local testing tool as well. It's worth noting that even though we were discussing OS X Server, Tomcat and other server-side application engines can be seamlessly executed on a regular OS X operating system as well; OS X Server just makes the step easier with an embedded installation/configuration. From Apple's perspective it would be nice to make Tomcat configuration/administration seamless from the server console, similar to the way it's done for other services such as Apache Web Server and FTP Server. TomcatX (http://homepage.mac.com/rlaing/TomcatX.html), a simple Cocoa-based graphical utility, was also useful in starting Tomcat instances.

Beyond Tomcat: J2EE

Since OS X completely supports J2SE 1.3, other third-party J2EE application servers such as Orion and JBoss, which are completely written on top of Java, can be used to develop and deploy J2EE applications on an OS X platform as well. For instance I successfully tested the latest version (1.5.2) of the Orion Application Server (which is also the key constituent of Oracle's J2EE application server) on my OS X Server installation. The Orion Application Server (www.orionserver.com) is a freely available (for development purposes) J2EE-based application server and implements key J2EE technologies including EJB, JNDI, JTA, JMS, JDBC, JSP, and servlets. Apple's own WebObjects platform, which supports a server-side enterprise Java platform, is expected to move toward J2EE compliance as well. This area needs greater focus from Apple and third-party application server developers (such as BEA and IBM) to make J2EE development and deployment smoother on OS X.

Is Mac OS X a Perfect Candidate for Java Development?

It's clear that Java and OS X are good for each other. Should OS X be your next Java development environment? In this section I'll explore how OS X maps some fundamental requirements from a developer's perspective and attempt to map the various tools and technologies around OS X and Java by discussing the key development requirements and key enablers in Apple Mac OS X.

Java Runtime and Development Kit (Including Compilers)

OS X embeds J2SE version 1.3.1 with the HotSpot Client VM.

Support for J2EE-Based Application Servers

Apache Jakarta Tomcat, which implements the JSP and Java servlets component of J2EE, is embedded into the OS X Server platform; for nonserver users Tomcat can be easily installed and configured as well.

thought inc.

www.thoughtinc.com

The notion of running full-featured applications embedded in a browser has created a whole slew of controversies – problems with browser JVMs, version conflict, memory requirements with Swing-based applications, and more.

Sun recently released a technology called Java Web Start, a fairly significant new addition to the Java platform. Java Web Start combines the server-based management aspect of a Web application for app delivery, but still provides a complete, secure, application-based runtime environment for full-featured applications, instead of the traditional applet sandbox. Hopefully, we'll see Java Web Start implementing the original Java vision for the distribution of interactive Java applications.

Other pure Java-based J2EE application servers such as the Orion Application Server and JBoss can be used on OS X as well.

Java Development Tools

Numerous development tools are available, for example:

- vi/emacs
- Apple Project Builder
- Borland JBuilder 5.0/6.0
- NetBeans 3.2 and Forte for Java 3.0
- Metrowerks CodeWarrior for Java
- jEdit
- MightyJ
- Other popular IDEs are expected to follow suit with OS X ports

Database Connectivity Using JDBC

Database connectivity is a crucial aspect of most enterprise application development. Since OS X completely supports J2SE 1.3, JDBC support is built-in. If your database vendor or a third-party supplier has a Type 4 pure Java JDBC driver, you can use that driver with JDBC for OS X. A port of MySQL, a popular database system in the Linux community, is available for Mac OS X (actually OS X Server includes it as part of the installation). Other RDBMS options on OS X include PostgreSQL and InstantDB. Since OS X has roots in BSD Unix, we can expect more database vendors to port their databases natively for the OS X environment.

UML Modeling Tools

As highlighted earlier in the article, ArgoUML, a Java-based UML, is available as a Java Web Start application.

Installation Tools

To aid in the development of installation scripts/programs for deploying Java-based applications, a number of options are available:

- Project Builder–based creation of clickable .app applications
- Installer VISE 7.x from MindVision Software
- Install Anywhere 4.5 from Zero G Software

Ease of Deployment

Key highlights that aid the deployment of Java applications include J2SE with a HotSpot Client VM, support for Java in Internet Explorer, and Java Web Start.

Web Browser with Java Support

Microsoft Internet Explorer and Netscape X 6.2 with the latest Java support are both available on OS X.

Office Productivity Tools

Microsoft has ported Microsoft Office for OS X including Word X, PowerPoint X, Excel X, and Entourage X (e-mail collaboration tool). Office is now available on OS X, as it seamlessly makes life easier for Java developers on the Windows platform since they also need productivity tools (documentation, presentations, spreadsheets, etc.) apart from Java development. OpenOffice, a port of the open source office toolset, is underway as well.

Hardware

Apple is known for great hardware. Not only from the outside but also from the inside. For instance, my choice for running OS X would be the sleek Titanium PowerBook G4 Notebook. Powered by a fast 66MHz PowerPC G4 processor, 48GB hard-disk drive, support for up to 1GB of RAM, a slot-loading CD-RW/DVD drive, built-in wireless LAN support through Airport, and an almost infinite expandability option with FireWire and USB support, the 1 in. thick and 5.3 lb portable notebook with 15.2 in. display has everything a developer could dream about. And you can always make your family and friends feel good by creating movies and DVDs using Apple's iMovie and iDVD software when you're not developing Java applications.

Conclusion

OS X marks a revolutionary step toward usability and ease of use for a productive and a user-friendly operating system. With its built-in support for the latest version of the Java platform and the key benefits it brings to the table, including the excellent hardware, Microsoft Office, and built-in wireless connectivity, OS X is definitely a candidate for your next operating system for building and running enterprise-class Java applications.

References

- *Apple Mac OS X:* www.apple.com/macosx/
- *OS X Developer Community:* http://developer.apple.com/macosx/index.html
- *Apple Java Developer Community:* http://developer.apple.com/java/
- *Java Web Start:* http://java.sun.com/products/javaweb-start/index.html
- *ArgoUML:* http://argouml.tigris.org
- *JBuilder on OS X:* www.borland.com/jbuilder/mac/
- *Mac OS X Server:* www.apple.com/macosx/server/
- *WebObjects:* www.apple.com/webobjects/

AUTHOR BIO

Hitesh Seth is chief technology evangelist for Silverline Technologies, a global e-business and mobile solutions consulting and integration services firm. He has extensive experience in the technologies associated with Internet application development. Hitesh received his bachelors degree from the Indian Institute of Technology Kanpur (IITK), India.

hitesh.seth@silverline.com

compoze
software

www.compoze.com/jdj

JASON R. BRIGGS J2ME EDITOR

A Star Alternative

A few months ago Alan mentioned that he had finally shifted to Star Office. As someone who has been using the software suite since Sun took it over, I applaud his decision to move away from that other office package. However, the shift doesn't come without a few challenges that can be quite annoying when you first make the move.

Challenges such as SO5.2 trying to take over your entire desktop (a feature that, luckily, can be bypassed, although not entirely), and that drawings in a Word document are occasionally decimated when imported into SO. Certain default shortcuts are different, and the SO way of doing things is slightly different than the Microsoft way. Not worse, just different.

After a while I decided that I don't miss MS Office. I still used it at work (day job, that is), and apart from grammar checking – the results of which are a bit unpredictable, in my opinion – I can't think of one MS Office feature that I wish Star had.

With Star Office 6, the irritating desktop override is finally going away, and the application has been split into separate components rather than an integrated word processor, spreadsheet, etc. I haven't personally upgraded yet because I found the earlier beta a little buggy on my laptop.

Another piece of software I don't miss is Internet Explorer. After trying various other browsers (including Netscape 6) I finally settled on Mozilla about six months ago, and it seems to improve with every new release. Performance is almost as good as Explorer (at the very least, after a few days you don't notice the difference any more). If some Web sites don't work with it, I just don't go back to them. You might be wondering what am I getting at here? Well, since Sun released a version of the J2ME Toolkit that ran on Linux, I realized there's no reason for me to be running a specific operating system anymore. Since I don't need Windows and I prefer Unix, then I might as well be running Linux (besides the fact that getting a winmodem working with Linux is a complete nightmare).

It's a bit of an eye-opener when you actually sit down and work out what you're really using your computer for, what's available for the other platforms, and exactly how much it's costing you to be on the cutting edge.

So what do I use my computer for? Surfing the Net (plus e-mail), writing, development, occasionally listening to music, and, very occasionally, playing some games. If we look at each of those in turn:

- ***Surfing the Net:*** Mozilla is my choice (available for Windows, Linux, and various other platforms). The e-mail client provided is almost perfect for my needs. The only problem I have with it is the message-rules system, which still seems buggy.
- ***Writing:*** Star Office, as mentioned, suits my minimal needs and is available for Windows, Linux, and Solaris.
- ***Development:*** The JDK is obviously available for numerous platforms. There's a wide variety of IDEs available for Java, so the OS is not likely to be a problem. At the very least Jext, a Java text editor, provides enough functionality for my purposes (www.jext.org). JBoss and Tomcat, my application and Web server of choice, will pretty much run on anything, and the MIDP emulator (J2ME Toolkit), as mentioned, runs on Linux, Windows, and Solaris.
- ***Music:*** MP3 players are available for any platform you care to name. Okay, there's no MP3 player for an abacus. All right, there's probably no MP3 player for your Texas Instruments calculator. Smart arse.
- ***Games:*** The idea of a games console used to be anathema to me, but then I looked at the price of the latest and greatest graphics card, plus remembered prior experiences with cutting-edge games not running well on my – at the time – cutting-edge computer. So I've changed my tune, hence, the next hardware purchase I make will probably be a PS2.

I recently worked out the price for a Linux desktop system, including a Playstation 2, for those moments when I need to wear my gaming fez (if that means nothing to you, see www.pvponline.com), and it came to around 65% of the price of the Wintel machine I would need for the same purposes; of course, this assumes I'd need a cutting-edge machine for Windows. Past experience with new versions of Windows tells me I would. Interestingly, the price of a new Mac, with the Unix-based OS X (plus a Playstation 2), is only slightly more expensive than the Windows box.

So why am I still running Windows? Anyone who has managed to get past the horrors of the Linux winmodem driver, only to get stuck with the joys of PPP, please contact a depressed *JDJ* editor. I'm starting a support group for laptop owners who are trapped in Windows hell to support their Internet addiction.

jasonbriggs@sys-con.com

AUTHOR BIO

Jason R. Briggs is a Java analyst programmer and – sometimes – architect. He's been officially developing in Java for almost four years "unofficially for five."

J 2 M E I N D E X

A Star Alternative 98
A few months ago Alan mentioned that he had finally shifted to Star Office. As someone who has been using the software suite since Sun took it over, I applaud his decision to move away from that other office package.
by Jason R. Briggs

Jini Surrogate as a Platform for J2ME Games 100
Surrogate architecture incorporates smaller devices
by William Swaney

Building End-to-End Palm Applications Using Java 106
A worthwhile and useful experience
by JP Morgenthal

Leveling the Playing Field 112
A J2ME co-op could let the little guy come out of the cold
by Jason R. Briggs

Integrating J2ME, GPS, and the Wireless Web 118
Developing location-based applications
by Shane Isbell

qualcomm incorporated

www.brew2002.com

Jini Surrogate as a Platform for J2ME Games

Surrogate architecture incorporates smaller devices

WRITTEN BY
WILLIAM SWANEY

Imagine using your J2ME device to participate in a complicated online game – or a simple one, for that matter. You log in to a network where network services are elements of the game. You, as a player in a massive online world, are represented as an object, a peer of all the other game elements.

Your player object becomes the client of a map service, a service that allows you to explore and move around while delivering necessary display information. Your object discovers and uses other services as needed, without prior knowledge of many of them. You meet a creature you've never encountered before – is it a computer-controlled character or another player?

The criteria for finding game elements can actually be tied to real-time game play. You may have access only to services with a notion of proximity to your object; maybe the map service is really a group of services representing multiple locales. As you move from location to location, you meet different people and find different items. All this interaction occurs with your player object, an object that is really your interface to this world, an object you control from your J2ME device. Unfortunately, a billing service finds you, recognizes that you're now playing, and starts to charge you accordingly.

Jini is an ideal platform for hosting games for J2ME devices. But what does "hosting games" mean? Jini is a technology concerned with network services: how services interact on a network, how they discover other services, how they handle failures, and how an ever-changing network topology is best managed. Why not view mobile devices as extensions of the network, part of a group of services that together perform the functions necessary for a game?

This article introduces the Jini surrogate architecture, along with J2ME devices, as a viable platform for games. I'll discuss the basic concepts in the hope of provoking thought on further possibilities. In addition, I'll show how to get Sun's reference implementation, Madison, up and running.

Perception is everything When Jini was introduced, it was perceived as a technology for connecting devices to a network. This proved difficult to implement, however, as the footprint Jini required was, and still is, rather large. To address this, a project was started at www.jini.org to develop a "surrogate" architecture to incorporate smaller devices. This architecture is what I propose as a platform for J2ME games.

What Is the Surrogate Architecture?

The surrogate project defines architecture that allows devices that normally wouldn't be able to participate in a Jini network to do so. This is done by bridging the device and its environment with one capable of interacting with the Jini network. To do this, an object (or surrogate) is created to act on behalf of the device. A device finds a surrogate host in its "native" network environment and registers with it, providing the surrogate host with either a JAR file or the location of a JAR file. The surrogate host instantiates a surrogate object, which is obtained from the JAR file. The surrogate object becomes the device's representative on the Jini network. Communication now happens separately from the surrogate host, as the surrogate object and the device are now responsible for their own communication.

The general architecture can be logically broken down into its components: a surrogate host and a connector specification. The former provides a context for the surrogate object, including an export service for distributable code, and the necessary lookup and discovery management for the Jini network. The connector specification is responsible for defining how the device discovers and registers with the host. Finally, the device and surrogate object communicate in a manner that may not be related at all to the connector used to register the device.

What a Surrogate Brings to the Table

There are several compelling reasons to use the surrogate architecture as a platform for J2ME games. One is that the architecture simplifies device management. The device presents itself to the Jini network through a reference to a surrogate object codebase. The capabilities of the device don't need to be determined by a back-end server; rather, they are defined within the surrogate object code itself. A present surrogate might be programmed for the Connected Limited Device Configuration. In the future, as different CDC devices become available, a surrogate specific to them can be programmed. These new surrogates join the network, just like the others, with their own inherent capabilities. The services already on the network don't need to know about the new device.

Wireless connections aren't noted for their stability. With surrogate architecture, if connections come and go, the surrogate object can remain active, maintaining information regarding the current session and playing an active role on behalf of the device in the Jini network. When the device reconnects, the channel to the surrogate object can be reestablished. Often this can be done invisibly to the device users; they may never know they were disconnected.

Resource handling becomes easier through the use of built-in Jini features such as leasing. The surrogate object and device are responsible for maintaining a keep-alive for their connection. Should the device lose the connection ashnasoft corporation

www.ashnasoft.com

altogether and not reconnect, the surrogate can clean itself up after a certain amount of time. In addition, the surrogate object may register itself with one or more lookup services (an LUS provides the functionality to discover and register services). This is a leased registration, and is therefore self-cleaning.

Mobile Devices as Unique Objects

Mobile devices are viewed as unique objects by the Jini network. Just as layers of complexity are added to the device's client software, layers can be added to the surrogate object (through interfaces, for example). This means that a game system can view the surrogate object as a player with various properties. One such property might be "I want to play chess." Another Jini service on the network might connect players based on the games they want to play.

Having devices as unique objects in the Jini network allows a richer set of interaction semantics. In a portal model, device login and registration may appear the same from the device's point of view, but the other services view the portal, not the individual objects. This places the responsibility for initiating service interaction on the portal. In the surrogate architecture model, other services can initiate interaction on many levels: to an individual surrogate object, to a group of surrogate objects meeting some criteria (interfaces implemented, or any other attribute), or simply as the entire group of surrogate objects.

You might have an adventure game with a "universe" service. This service might represent the map or playing field for the game and may wish to find players based on a virtual locale. The locale of the player could be kept within the surrogate object's Jini registration criteria.

Getting Started

Before we start, if you haven't already done so, you'll need to download and install Jini and the Madison project. Jini can be downloaded from www.sun.com/jini/; Madison can be found at http://developer.jini.org/exchange/projects/surrogate/IP/. A free membership at the Jini.org site is required for downloading.

Here are the parameters that should be set in "ENVIRONMENT.bat":

```
set JAVAHOME=E:\jdk1.3\bin
set JINIHOME=E:\jini1_1
set MADISONHOME=E:\jini_surr_madi
  son1_0
set GROUP=the2bears
set HOST=frodo
set JPORT=8080
set MPORT=8083
```

First Jini...

Before starting Madison we need to have a basic Jini network running. I won't go into Jini in depth, as it's beyond the scope of this article; in addition, there are many good tutorials and books out there.

We'll need three basic things to start: the rmid activation server, an HTTP server for serving classes, and an LUS. A fourth application, an LUS browser, will be started to give us a visual feedback of what is happening in our Jini network.

The LUS requires an RMI activation daemon. Start this using the following, which may be in a batch file:

```
call ENVIRONMENT.bat
%JAVAHOME%\rmid -J-Djava.security.pol-
icy=%MADISONHOME%\policy\policy.all -
J-Dsun.rmi.activation.execPolicy=none
-log .\rmid
```

An HTTP server is also necessary because we'll be downloading code to different clients in our Jini network. Jini is a specification, and we'll use Sun's reference implementation. It's reasonable for a client of an LUS to know the inter-

aquarius
solutions

www.aquariussolutions.com

softwired, inc.

www.softwired-inc.com

face, but not necessarily the implementation. With an HTTP server downloading classes, the client of a service can get the classes it needs. Start the HTTP server with:

```
call ENVIRONMENT.bat
%JAVAHOME%\java -cp
%JINIHOME%\lib\tools.jar
com.sun.jini.tool.ClassServer -port
%JPORT% -dir %JINIHOME%\lib -trees -
verbose
```

The LUS – in our case Sun's "Reggie" – is the bootstrap service for a Jini network. All other services need somewhere to register their proxies and to find the proxies for other services. An LUS starts up and registers its proxy with itself. Start the LUS with the following:

```
call ENVIRONMENT.bat
%JAVAHOME%\java -jar
%JINIHOME%\lib\reggie.jar
http://%HOST%:%JPORT%/reggie-dl.jar
%MADISONHOME%\policy\policy.all .\reg-
gie %GROUP%
```

The LUS browser provides a simple interface that allows us to select the service registrar we wish to view and the services registered with that registrar. Upon starting the browser, you may notice that the HTTP server (if started in "verbose" mode) logs a request for the reggie-dl.jar file. This is the code necessary for the proxy to the service registrar. The LUS browser can be started with:

```
call ENVIRONMENT.bat
%JAVAHOME%\java -cp
%JINIHOME%\lib\jini-examples.jar  -
Djava.security.policy=%MADISONHOME%\
policy\policy.all -
Djava.rmi.server.codebase=http://%HOST
%:%JPORT%/jini-examples-dl.jar
com.sun.jini.example.browser.Browser -
admin
```

Note that the order in which the files are executed is important. Each of these files references the file ENVIRONMENT.bat, which must be edited to reflect your system.

...Then Madison

Madison includes three pieces for us to start: Madison itself (the surrogate host), a simulated device that implements the IPConnector protocol, and a test client for the device. Properties for Madison are set in the file "madison.prop" (see Listing 1), which is included in the Madison download.

Madison doesn't register as a Jini service upon startup, but it requires the Jini network to be running. However, part of its responsibility is to provide a context for surrogate objects so they can perform Jini lookup and discovery. In addition, Madison provides an export service that a surrogate object registering in an LUS must use for its code to be downloadable within the Jini network. Finally, Madison starts the IPConnector protocol. Start Madison with this script:

```
call ENVIRONMENT.bat
%JAVAHOME%\java -Djava.security.poli-
cy=%MADISONHOME%\policy\policy.all -
jar %MADISONHOME%\lib\madison-boot.jar
-prop %MADISONHOME%\bin\madison.prop
```

The device simulator included in the Madison project is actually a small application that makes it easy to test surrogate objects. In the simulated-device.prop file (see Listing 2) we have specified a surrogate object to register. The device simulator discovers Madison and registers the surrogate object using the IPConnector. If registration is successful, the surrogate object will now appear in the LUS. The surrogate object will register itself with the LUS, so its code must be downloadable. We'll use a second HTTP server for this. Start the device simulator and the second HTTP server with these two scripts:

```
call ENVIRONMENT.bat
%JAVAHOME%\java -cp
%JINIHOME%\lib\tools.jar
com.sun.jini.tool.ClassServer -port
%MPORT% -dir %MADISONHOME%\lib -trees
-verbose
```

and

```
call ENVIRONMENT.bat
%JAVAHOME%\java -jar
%MADISONHOME%\lib\madison-device.jar -
prop simulated-device.prop
```

Finally, we start a test client for the device. The test client, upon startup, obtains a reference to the first surrogate and allows us to make a remote call against the surrogate's proxy – in this case obtaining the description. Start the client for the device:

```
call ENVIRONMENT.bat
%JAVAHOME%\java -Djava.security.poli-
cy=%MADISONHOME%\policy\policy.all -
Dcom.sun.jini.madison.debug=true -
Djava.rmi.server.codebase=http://%HOST
%:%MPORT%/madison-client-dl.jar -jar
%MADISONHOME%\lib\madison-client.jar -
groups %GROUP%
```

The files for starting Madison are:

- lstartMadison.bat
- lstartDeviceWebServer.bat
- lstartDevice.bat
- lstartDeviceClient.bat

Two additional files must be edited: madison.prop and simulateddevice .prop. Specifically, there are classpath properties and codebase URL properties that must reflect your system.

Future Path

The system as we now have it consists of a surrogate host running, with an IPConnector announcing and listening for registrations. A device simulator sees the host's announcement and registers a surrogate with it. This surrogate, using the context provided by the host, is able to register itself with the Jini network, and clients of the surrogate can invoke remote methods.

So far, though, we haven't connected a mobile device to the Jini network. Since our goal is to use J2ME devices with a Jini network for games, we'll need to define a connector to the surrogate host that they can use. Once we accomplish this, we have an effective framework for device-to-surrogate communication and we can start to build a game on top of this.

william.swaney@valaran.com

AUTHOR BIO
William Swaney, a software developer specializing in distributed computing, works for Valaran Corp., where he experiments with mobile devices and Jini networks.

Listing 1: madison.prop

```
com.sun.jini.madison.commonCLPath=/E:/jini1_1/lib/jini-
core.jar;/E:/jini1_1/lib/jini-ext.jar;/E:/jini_surr_madison1_0/lib/surro-
gate.jar;/E:/jini_surr_madison1_0/lib/IPinterconnect.jar
com.sun.jini.madison.hostCLPath=/E:/jini1_1/lib/sun-
util.jar;/E:/jini_surr_madison1_0/lib/madison-impl.jar
com.sun.jini.madison.DiscoveryManagement.groups=the2bears
```

Listing 2: simulated-device.prop

```
com.sun.jini.madison.debug=true
surrogate.0=http://frodo:8083/madison-surrogate.jar
description.0=First Surrogate (Surrogate 0)
```

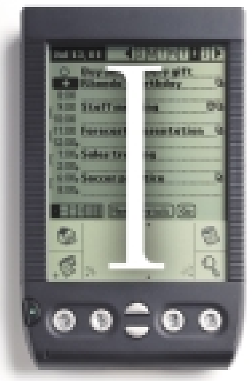
Download the Code!
www.JavaDevelopersJournal.com

borland
software corp.

www.borland.com/new/
optimizeit/94000.html

Building End-to-End Palm Applications Using Java

written by JP Morgenthal



In the past, mobile warriors were the only ones who relied on portable information technology. Since PalmOS, RIM, and WindowsCE devices penetrated corporate walls, it's no longer unusual to have over 60% of corporate employees using PDAs and handheld devices for time management.

Indeed, Franklin-Covey, one of the world's largest providers of time-management tools, adopted the medium and made it a large part of their overall toolkit.

As with all great technologies that improve productivity, it's taken the enterprise quite a bit of time to catch up and analyze what's happening right under their noses. So employees have fortified themselves with the ultimate arsenal for the dissemination of corporate information. Now comes the real trick for corporate IT departments – how to build and deploy applications on PDA/handheld platforms that support management's goal of providing employees with corporate information when they're away from their desktops.

This will truly be one of the most complex endeavors for IT departments (since the proliferation of desktop operating systems). There are three versions of Palm operating systems, two of RimOS, and three of WindowsCE. Each version has varying capabilities and functions. In addition, Palm has licensed their operating system to other vendors who have added their own capabilities for their own devices. Kyrocera and Handspring are two examples.

The industry faced a similar situation when Java first arrived on the scene with a solution for cross-platform application development. Well, once again Java is offering an opportunity for cross-platform application development on PDAs and handheld computers.

The Java 2 Micro Edition (J2ME) is a Java 2 platform used in small footprint devices, such as PDAs, phones, and appliances. The goal of J2ME is to separate the set of usable APIs into groups based upon device functionality. These groups are defined as "profiles" and allow applications to be designed for specific sets of capabilities.

There's basically one choice for developers who want to build cross-platform applications for RIM and PalmOS devices: the Mobile Information Device Profile (MIDP). I doubt anyone reading this article would expect to find a WindowsCE solution here, although I won't rule out that an implementation of MIDP for WindowsCE might already exist.

The Application

After downloading and installing the sample applications that come with the MIDP implementation for PalmOS from

fiorano software

www.fiorano.com

Sun's Web site (http://java.sun.com/products/j2mewtoolkit/), I decided to build a J2ME application (MIDlet). This entailed studying the MIDP specification, which provided some basic direction but also required significant testing with the individual platforms to really understand how they would impact the user.

The application I chose to build is called SecurePad. I chose it because I hate locking my Palm (since it makes it a real pain to use), but I don't want confidential information in the open if I should lose my device. SecurePad requires the user to enter a password before being placed in a memo pad list screen. That password is then used to encrypt/decrypt all messages for that session. This approach yielded some interesting side effects, as it allowed me to have a different password for each note if I so chose.

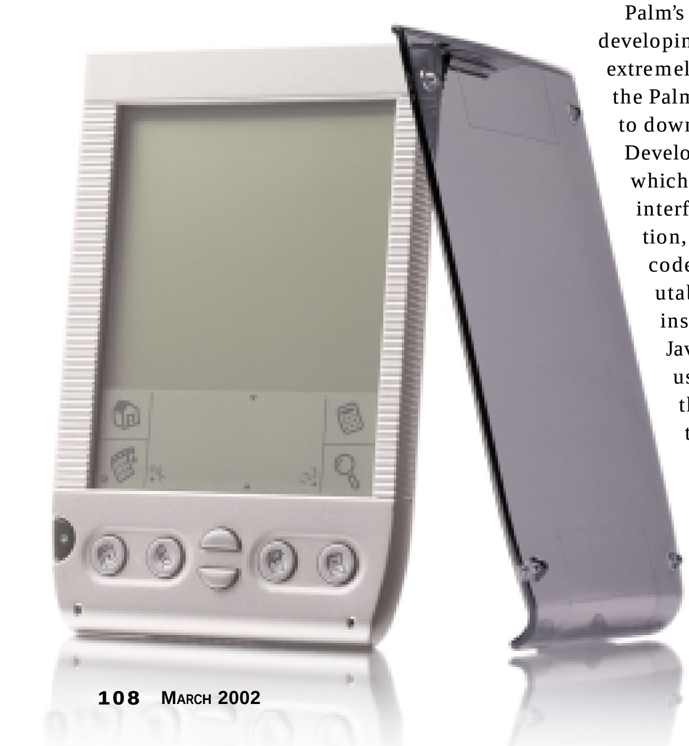
The key to SecurePad is the integration with Palm conduits, which allows the entire pad to be uploaded from or downloaded to the desktop without first being decrypted. This is a critical part of building an end-to-end Palm application as it's possible for the Palm to corrupt data or need a hard reset, which would erase everything in memory.

The Results

J2ME applications definitely don't look or feel like native PalmOS applications. After years of hearing that Java applications don't look and feel like Windows applications, this should come as no surprise to Java developers. My biggest complaint was that all field-based input had to be provided through the use of a Form object, which requires the form to occupy the entire screen. And there are only a small, fixed number of items to choose from for screen objects, but the basics such as lists, buttons, and images are there.

Note: The current MIDP version used for the Palm also needs to operate on cellular phones and similar types of equipment that have a lot less screen space and a very mundane entry system. The Sun community is working on a different profile that fits a much more powerful class of devices.

SecurePad uses a basic XOR encryption scheme and operates directly on byte arrays. Still, this modest amount of memory access and mathematical computation appears to make my Visor Prism seem like an IBM PC XT loading up Lotus 123 off a floppy disk. I've since been testing with a C++ version of the application and the encryption/decryption algorithm works considerably faster than under Java.



Palm's Java interface for developing conduits is an extremely useful tool. From the Palm Web site I was able to download the Conduit Development Kit (CDK), which includes the Java interface, documentation, usable sample code, and a redistributable component for installing and using Java conduits on users' desktops. All things considered, it took only a few hours to transform their TEXTCOND sample application into a usable conduit for SecurePad (see Figures 1–3).

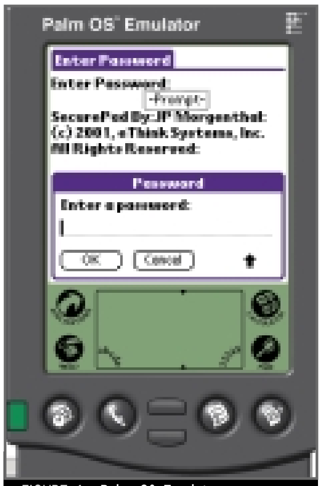
The How-To

To build an end-to-end application for the PalmOS in Java you'll need the following items:

1. ***The PalmOS emulator and a PalmOS ROM file:*** If you have a serially connected base, you can download the ROM from your own Palm device. If you have a USB-connected base, it may be difficult to download the ROM from your Palm to your desktop.
2. ***The J2ME Wireless Toolkit:*** The toolkit ships with emulators for cellular phones and can integrate into the PalmOS emulator.
3. ***The Java VM for PalmOS 1.0:*** This VM conforms to the MIDP and you'll need to install this on your Palm before you can load and run MIDlets.
4. ***The Palm Conduit Development Kit 4.0***

The J2ME Wireless Toolkit (J2MEWTK) supports Sun's Forte development tool; however, I was unable to get the complete environment working and found it easier to use Sun's KToolbar application that ships with the J2ME Wireless Toolkit. The KToolbar application performs the necessary builds of the application and creates the .JAD manifest file for the MIDlet.

The J2MEWTK also comes with tools to transform .JAD and .JAR files into .PRC files for uploading into the Palm. An automated tool comes with a graphical user interface that can be launched though the J2MEWTK utilities. However, this application frequently failed and I ended up generating the .PRC from the command line. This is a very important point if you're using a conduit with your application, since the creator ID on the Palm application is used to identify which conduit to call upon synchronization. The command-line application is the only way to apply a creator ID to your MIDlet. We'll review this further when discussing how to build the end-to-end application.



Enter Password:
SecurePad By: JP Morgenthal;
(c) 2001, eThink Systems, Inc.
All Rights Reserved.
Enter a password:

FIGURE 1 Palm OS Emulator

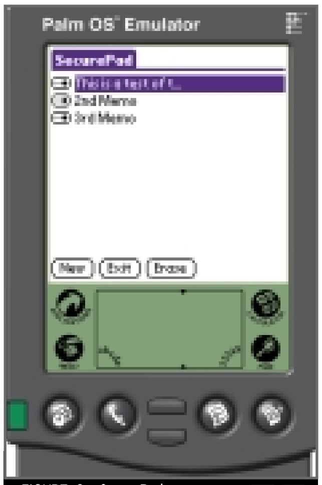
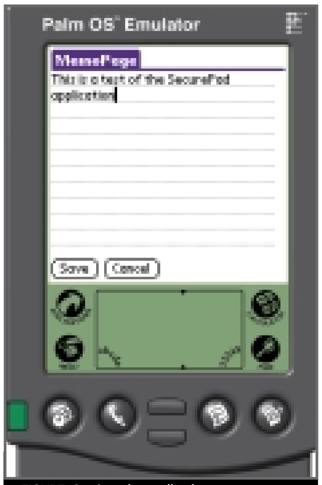


FIGURE 2 SecurePad

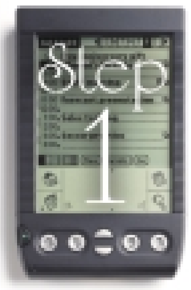


This is a test of the SecurePad application

FIGURE 3 Sample application

inetsoft
technology corp.

www.inetsoft.com



Developing the application

A key thing to know about designing your J2ME application for the PalmOS is that when the application exits, it's over. As you read through the documentation you'll see that the MIDP profile supports a paused state for the application. This initially led me to believe that the Palm application would run in the background and, therefore, I designed the first incarnation of SecurePad around that thought.

Now, having delved a little deeper into the workings of the Palm, I understand that it's the responsibility of the application to store and retrieve its state upon exit and reentry. However, J2ME developers will need to use a database record to make this happen because MIDP doesn't provide access to the properties database on the Palm, which is where this information would typically be stored.

All MIDlets start by extending the javax.microedition.midlet.MIDlet class. This enables the Palm to launch the application. Once launched, it's up to the developer to set the current screen. Building a MIDlet reminded me of the Macintosh Hypercard programming motif – essentially, screens are stacked and unstacked with the current screen controlling which screen should be next. This is only one model; the user could have the controlling class make all decisions on screen based on state information.

If the application requires persistent storage on the device, the MIDP provides the RecordStore, a nice abstraction of the underlying database mechanics of the Palm. It's a fairly good representation of the Palm's native database management capabilities and is extremely easy to use. However, just like JDBC, remember to close your database before exiting the application. I witnessed some strange behavior on the emulator when previous instances didn't close the RecordStore.

Listing 1 illustrates how to create a new form and read records from the database. (Listings 1–2 can be downloaded from www.sys-con.com/java/sourcec.cfm.)

For debugging I used the phone emulators during development stages since they leveraged KToolbar's console to deliver messages sent to System.out and System.err. The Palm emulator won't perform this function, making debugging very difficult. To debug the few things on the Palm that I had to, I built Alert dialogs and displayed them.

Preparing the application for deployment

After the application is built and tested, the KToolbar application will generate a .JAD and .JAR file for you. These files will be stored in the application's bin directory.

To synchronize this application with your Palm, create a .PRC file. The J2MEWTK contains a utility in the C:\J2mewtk\wtklib\devices\PalmOS_Device directory to transform .JAR files into .PRC files. The following command performs this task for you:

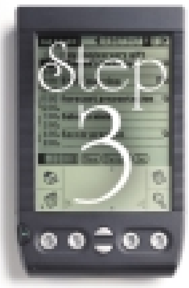
```
java -jar
C:\J2mewtk\wtklib\devices\PalmOS_Device\MakeMIDPApp.jar -
creator [creatorID]  \
-JARtoPRC [JAR file] [main class]
```

For example:

```
java -jar
C:\J2mewtk\wtklib\devices\PalmOS_Device\MakeMIDPApp.jar -
creator spad  \
-JARtoPRC SecurePad.jar com.comtellect.securepad.SecurePad
```

(Don't enter the \ in the above command, it's just there to illustrate that the command is entered on a single line.)

Designing and building a conduit in Java



The CDK allows developers to use the Java programming language to build applications that will synchronize data with the Palm. This is a very powerful tool that has a proprietary API and allows full use of the J2SE environment.

There are two distinct components to building the conduit:

1. Installing the conduit into the HotSync Manager environment
2. Accessing data on the Palm during a HotSync operation

There's an API that allows developers to call the HotSync Manager functions for registration and removal. The CDK comes with a graphical utility for Windows called CondCfg.exe. This program lists all the current conduits registered and allows you to add and change their details.

Conduits have two entry points, one for configuration and one for execution. When the HotSync Manager locates an application database with a matching creator ID in its registry, it executes the conduit associated with that ID. Upon execution, the HotSync Manager calls the open() function on your conduit. Pay attention to the direction of the synchronization as it's passed in. This parameter is set by the user through the HotSync Manager if your conduit allows configuration.

The SyncManager object enables the conduit developer to read and write database records directly inside the Palm. However, the conduit must know the format of the database records in order to operate over them. Listing 2 illustrates uploading or downloading information on the Palm.

Conclusion

RIM will be shipping a fully functional version of their MIDP implementation, providing developers with a wider array of platforms to deliver their J2ME apps to. Eventually, both RIM and PalmOS will hopefully gain more powerful profiles based on the CDC. MIDP is now based on the CLDC.

While writing this article I quickly became aware of how much is involved in building an end-to-end application for the PalmOS. In the end, I focused on some key points I thought would help speed developers on their way and covered some of the hurdles I encountered to get my application to run. Overall, the experience of building an end-to-end Palm application in Java was a worthwhile and useful experience.

AUTHOR BIO
JP Morgenthal is CTO for Ikimbo and an expert on the design and implementation of distributed systems for the enterprise, and underlying technologies: Java, XML, enterprise application integration (EAI), and, business-to-business (B2B).

jpm@ikimbo.com

sun solutions

www.solutionssite.com

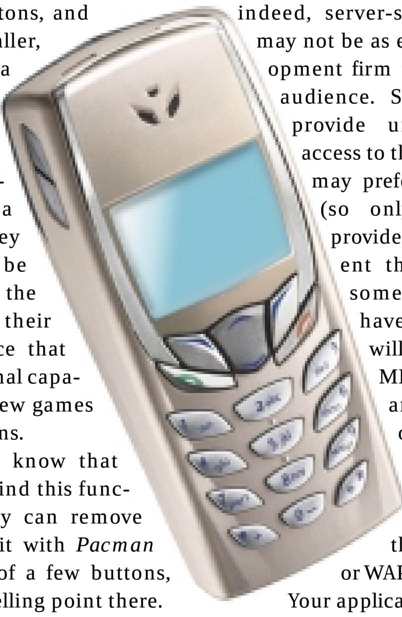
Leveling the Playing Field

A J2ME co-op could let the little guy come out of the cold

WRITTEN BY
JASON R. BRIGGS

You've heard this said before. In fact, if you regularly peruse the pages of *JDJ*, you've heard it here more than once. In case it hasn't sunk in, repeat after me: J2ME (especially MIDP) will provide tremendous opportunities for developers.

Not convinced? Think about it, then: a virtually untapped market of mobile-savvy users who aren't likely to dish out the thousands necessary to buy a computer, but will probably be willing to hand over a lot less money for something like a mobile phone. There are a lot of computer users out there, to be sure, but there are a lot more mobile phone and handheld device users. And mobile phones are far less daunting for a nontechnical person to come to grips with. There are fewer buttons, and the screen is smaller, which necessitates a much simpler user interface. It's a technology that people are already comfortable with – hence it's a good bet that they aren't going to be uncomfortable with the idea of replacing their phone with a device that provides the additional capability to download new games and other applications.



They might not know that Java is the force behind this functionality, but if they can remove *Snake* and replace it with *Pacman* with just the click of a few buttons, there's definitely a selling point there.

Stumbling Block

The one stumbling block in Java's bid for world domination is the massive glut of mobile phones already on the market. People will undoubtedly need to upgrade at some point – as their phones get lost, stolen, broken, or become unfashionable – but how long will the process actually take before the majority of phones in use are MIDP-enabled? Well, I guess that's for the phone manufacturers to predict.

A Veritable Obstacle Course

While MIDP development presents numerous opportunities to developers, the restrictions inherent in the platform also place a number of fairly major obstacles in your path (see Glen Cordrey's article "Pushing the Limits," *JDJ*, Vol. 6, issue 12).

Unlike when developing applets (or, indeed, server-side applications), it may not be as easy for a small development firm to reach its intended audience. Some networks may provide unrestricted Internet access to their customers; others may prefer a portal approach (so only a few preferred providers will be able to present their wares). Perhaps some mobile users who have a home computer will come across your MIDlet suite on the Web and choose to load it onto their phone. Users with no computer may have only restricted access through a TV, console, or WAP phone.

Your application may require various utility/support classes to provide necessary functionality; considering the memory on a mobile phone available for a MIDlet suite, if every suite uses the same library, a lot of storage space is wasted for no good reason.

Assuming you want to make money from your development efforts, how do you charge? Who do you charge? It will be beyond the resources of many small developers to go it alone, negotiate a contract with Sprint (for example), and interface with their back-end systems so that:

1. The software can easily be provided, in the right way, to the right phone.
2. They can charge the end user for the privilege of using it. (Unless, of course, they're selling their software outright – and possibly selling away a chance at larger profits).

These difficulties mean that we as developers will have to come up with alternatives to the prepackaged shareware and e-commerce methodologies that have provided revenue streams in the past.

An Almost Open-Source Portal — the MIDP Cooperative

A number of companies have already developed (or are in the process of developing) server-side applications that assist in delivering applications to mobile phone users, along with the associated tasks involved in the process. *Mobile provisioning* is one of the terms used to describe these kinds of applications. Most provisioning applications will require a considerable investment (financial, infrastructure, and otherwise), and certainly the target market will be big corporations. Nextel, as an example, could use a provisioner to handle the applications available for their MIDP-capable phones.

Small developers, however, could be left out in the cold. Suppose I produce a game that I'd like to make money from; currently, I'll have to contact the Sprints, Nextels, and BTs of every country in which I want to sell it. Then I'll have to do a good enough sales job to convince them to either buy my software outright or give me a share of the revenue when their customers play my game. There

pramati
technologies
www.pramati.com

aren't a lot of other options. So...enter the MIDP Co-op.

Safety in Numbers

At www.midlet.org you can find a free Java repository for downloading various applications to your mobile phone. You can either download apps to your PC (and then transfer them to your phone from there) or use OTA (over-the-air) downloads if your network and phone provide that capability.

The MIDP Co-op (yes, it's a crappy name...no, it wouldn't actually be called that) would work in a similar way. However, instead of downloading any number of suites (usually one application per suite in the case of midlet.org), users could select a number of applications that would be packaged into a single suite, ready for download.

Rather than individual developers having to charge for their applications, the co-op would do the job. Users who want to download software to a PC use traditional e-commerce mechanisms (a secure Web site with some form of merchant/shopping cart facility), but OTA would require the co-op to negotiate with the various networks over delivery and payment. In the best tradition of the middleman, a certain percentage of the sale (or use fee) of an application would go to the co-op. Most of the money, of course, should (and would) go to the developer.

Here are a few more ideas on how it might work:

Compiled at Our Place

In the co-op, as final compilation would potentially be done on the server, only authorized utility libraries could be used in a hosted application. For example, applications requiring a floating point class would have to use the class provided by the community. If some guy decides that his floating point class is better (more efficient perhaps), he can submit documentary proof to the board of directors, who would make the final decision (probably based on its impact on other hosted applications). All utility classes would be required to be open source to community members, and, in the interest of fairness, developers of utility classes should be allocated a percentage of any sales.

Philosophy Be Damned

To save on philosophical arguments, it should be up to the individual developers whether their application source would be made available to other developers in the community – hence "almost" open source. The source would, of course, have to be uploaded to the server for any application to be hosted (and compiled) on the server, but it could be viewed only by community members, and only if the developer allows. Obviously, open sourcing an app should be encouraged because it means that the community as a whole benefits (newbie developers looking at veteran developers' source code to determine the best way to accomplish a task, for example), but "religious views" should not be forced upon the members. One way of looking at it: If I don't want to open source my application, then why the hell should I?

Remember that there's a benefit to be gained in either case. Whether closed or open source, the more applications a co-op hosts, the more likely it is that visitors will come, download, and spend their money!

Multiple Personalities

Initially the co-op would host only client applications. At some point it might possibly become financially viable to set up full server-side hosting as well. This means multiuser (multiplayer) applications with application server and database access. Multiplayer games seem to be a big draw in the PC world, so it stands to reason that the same will be true in the phone market – hosting multiplayer apps is going to be necessary for the co-op to thrive.

Tell It to the Board

The board of directors should initially be drawn from "interested" parties, those who have contributed to the estimated running costs (e.g., hosting, legal expenses) for the first years of operation. If/when the co-op becomes profitable, directors would be repaid (with interest, based on typical bank savings account rates over that period), and their position could then be voted on by the community at the end of the term of directorship (only community members would be eligible to run). If only contributors (those who have applications hosted on the site) could be community members, after the first elections only contributors would remain on the board.

Show Me the Money!

To get a better idea of how a co-op might work in the real world, let's look at some figures. To start with we'll make some guesstimates:

1. We'd have between 50 and 100 hosted applications at launch; midlet.org has, at the time of this writing, over 140 applications, so it seems reasonable to assume that the co-op could attract at least 50 for its initial launch (especially considering the enticement of possibly making money from your applications). Over time that number could be expected to grow, so the hosting requirements should take into account a certain amount of expansion.

2. Developers would set the price of their applications, but should be encouraged to seriously discount them for at least the first year of operation. (*Note:* Nextel is currently selling applications ranging from a few dollars to $12.95.) A certain percentage of applications would inevitably be free, which is a good thing if the co-op wants to attract more business.

3. The worst-case scenario for an application size is 128K, based on the maximum memory size in the MIDP specification. Assuming such a scenario, the storage space for a thousand MIDlets (if the number of applications hosted grew con-

Price of App (US$)	Percentage of Total Number of Apps	Number of Apps	Number in Average Suite	Price of Component of Average Suite (US$)
3.00	10%	10	1	3.00
2.00	10%	10	1	2.00
1.00	20%	20	2	2.00
0.50	30%	30	3	1.50
0.00	30%	30	3	0.00
TOTAL	100%	100	10	8.50

Table 1 Applications and prices

sharp

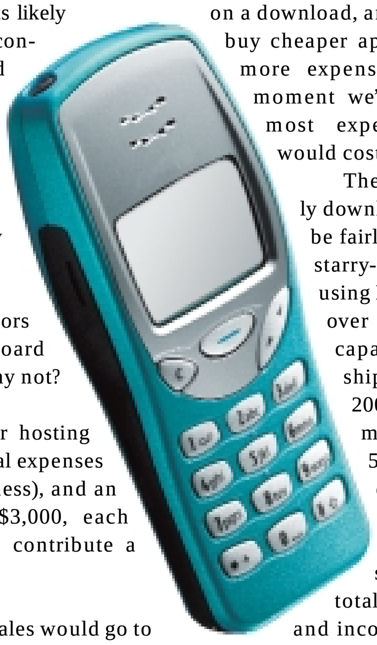
http: / / developer.sharpsec.com

siderably, of course) would be in the area of 256MB (assuming the co-op holds both source and compiled files). Adding various infrastructure components (an e-commerce initiative) to this figure, along with annual hosting costs likely to be at least $3,000 (a conservative estimate based on an extremely quick skim of some popular hosting companies), source control, community support functionality…and the figure rapidly starts to climb.

4. The number of directors on our hypothetical board would be 12. Why 12? Why not?

5. Assuming a two-year hosting cost of $6,000, initial legal expenses of $3,000 (a complete guess), and an emergency buffer of $3,000, each director would need to contribute a total of $1,000.

6. Ten percent of total sales would go to the co-op to cover running costs, expense payment, and perhaps a nominal fee to each director. Another 2% would go into a kitty to be paid to the utility class authors.



We'll make some additional assumptions regarding the average download and the distribution of applications according to price. At a guess, consumers wouldn't spend more than $10 on a download, and are more likely to buy cheaper applications than the more expensive ones. For the moment we'll assume that the most expensive application would cost $3 (see Table 1).

The number of monthly downloads would initially be fairly small, but let's get starry-eyed for a moment using Nokia's projection of over 50 million Java-capable mobile phones shipped by the end of 2002. Assuming only a measly 0.1% of those 50 million would download the aveage application suite in a year, that's still 50,000 people, total sales of $425,000, and income to the co-op (at 10%) of $42,500. The co-op could then consider purchasing its own server rather than using prepackaged hosting facilities, and paying off the initial board (even a minimal 5,000 downloads in a single year would mean that hosting expenses can be paid the following year).

The top developer might hope to attract 10% of the 50,000 downloads. Assuming the top developer also charges a top price ($3), he or she could make in the neighborhood of $15,000 for the year. Not a lot of money admittedly, but not bad, considering the developer would dish out a grand total of $0 to have the application hosted.

Community

This is only a simple sketch of what might go into one such co-op or foundation. One thing is certain: going it alone in the wireless environment would be extremely difficult for the average individual developer. As a consequence, the J2ME developer community will somehow have to come up with ideas that are acceptable to larger groups of developers so the little guy, not just the big corporations, can make some money from J2ME development.

After all, it's to be expected that some of the most innovative products will be those developed outside the confines of typically restrictive corporate environments.

Vive la communauté!

AUTHOR BIO

Jason Briggs is a Java analyst programmer and sometime architect. He's been officially developing in Java for almost four years, unofficially for five.

jasonbriggs@sys-con.com

solutionssite

www.solutionssite.com

northwoods software corporation

www.nwoods.com

actuate corporation

www.actuate.com/info/jbjad.asp

Targeting GPS

Integrating J2ME, GPS, and the Wireless Web

Written by **Shane Isbell**

For location-based services, the open frameworks of J2ME and J2EE create interesting opportunities in the fields of software development and applied statistics. Traditionally, the software industry in these services has been closed and, as a result, the industry has suffered stagnation, particularly in the area of distributed systems and integration.

Just look at this most recent example – U.S. cell phone carriers didn't meet the FCC October 2001 mandate for automatic location-based tracking for 911 calls over their networks. The most common reasons the carriers gave for missing the deadline were high costs and an inability to install the network infrastructure. With J2ME, XML, J2EE, and GPS, you can use the existing infrastructure (the Internet and your computer) to build and run such services from your garage, all for a very low cost (free).

With Java's open architecture, you can build more exciting applications. GPS data contains the altitude and speed of a target, allowing the extraction of 3D information and vector coordinates. This can lead to interesting implementations, such as topographic position tracking of multiple targets or predicting the time two targets will intersect.

The use of wireless GPS and J2ME also allows more accurate tracking of targets because the historical GPS data can be stored in a database, enabling the integration of stochastic models and applications to monitor, predict, and correct movements before sending this information to a mobile device. Because of the limited graphical capability of mobile devices, there's also a need for applications to convey location-based information through the use of colors, a feature of the more recent mobile devices.

The GPS application covered in this article tracks the user on a graphical plot and gives the distance of another target connected to the network. The target can also track the user. By going through the application, you can build a foundation to develop your own GPS services. We'll discuss how to set up the test environment and show how to parse and handle GPS data on a mobile device and transfer GPS information back and forth between the device and a J2EE server. Thus, by integrating J2EE and J2ME, the mobile device harnesses the power of the server, allowing the development of intelligent, location-based applications.

Setting Up the Test Environment

Setting Up a GPS Receiver

A GPS receiver must meet two requirements for the application to work. First, it should be able to output NMEA 0183 (version 2.0 or higher) data as text; second, it shouldn't require the receipt of an initialization string before sending data. eTrex from Garmin meets both requirements. Since the receiver hooks into the serial port on a PC (desktop or laptop), you need to obtain a GPS-to-serial-port adapter. Once connected, GPS data will flow through the communication port. Keep in mind that most GPS receivers don't work indoors, so place the PC near a window where the receiver has clear access to the open sky. The receiver will need to get signals from at least three satellites to determine its position.

For those who don't wish to go through the trouble and expense of dealing with a GPS receiver, the source code for this article includes a Java class file (HttpReader) that functions as a GPS data stream. (Listings 1–3 and the source code for this article can be downloaded from the ***JDJ*** Web site, www.sys-con.com/java/sourcec.cfm.) This class instance reads an HTML page that contains actual GPS data and returns its input stream. While this method is easier than using a GPS receiver, watching a plot of the author driving to his local grocery store is not nearly as much fun.

HTTP Connection

Since MIDP requires the implementation of a subset of HTTP 1.1, this protocol should work for all implementations of J2ME. Therefore the application uses HTTP to communicate with the server.

The server may reside on the local machine or on a LAN. For the server to be accessible over the Web, you need a wired or wireless Internet connection. However, a wireless connection provides the most usability for testing in a real production environment. This may be set up in a number of ways. A mobile phone with Internet access can hook up through a connector to a port on the laptop. Obtaining a wireless modem or doing a direct dial-up to a server hosting the server-side GPS application are other options.

mongoose technology

www.portalstudio.com

Software Requirements

The GPS application requires the J2ME Wireless Toolkit from Sun and JDK 1.3 or higher. You also need the kXML package from Enhydra to handle XML on the MIDP device, as well as the xspsoft class files that are included in the download.

For the server-side applications, this article uses JBoss/Tomcat J2EE, which can be freely downloaded from www.jboss.org. Of course, you can also use any commercial J2EE implementation. You also need JAXB and JDK 1.4 from Sun.

The server application allows the user to transmit the GPS coordinates and to retrieve other people's coordinates. If the you don't wish to bother with EJBs and don't care about tracking multiple targets, the server-side application can be eliminated without affecting the core GPS functionality on the MIDP device or emulator. (The application could also be rewritten using JSP and JDBC.)

NMEA Format

In the 1980s the National Marine Electronics Association (NMEA) developed the NMEA 0183 Interface Standard for data exchange between marine electronic devices. Today, most global positioning systems use the NMEA interface for data exchange. Thus the J2ME GPS data parser will work on any GPS receiver that implements this standard. Figure 1 contains a sample of a data stream:

NMEA Sentences

```
$GPRMC,214434,A,3753.666,N,12203.162,W,0.0,0.0,270901,15.4,E
,A*33
$GPGGA,214616,3753.667,N,12203.167,W,1,04,5.6,121.1,M,-
27.4,M,,*79
$GPGSA,A,3,01,03,20,22,,,,,,,,,7.4,5.6,1.5*36
$GPGSV,3,1,10,01,69,062,47,03,12,106,37,04,12,279,00,08,12,250,
00*77
$GPGLL,3753.667,N,12203.167,W,214616,A,A*54
$GPBOD,,T,,M,,*47
```

FIGURE 1: GPS data stream

The $GPGGA sentence contains the target's topological location information. The relevant fields are as follows:

```
 Field
1 Message Header ($GPGGA)
3 - 4 Latitude, North/South
5 - 6 Longitude, East/West
10Altitude in meters
```

The latitude 3753.667 denotes 37 degrees, 53 minutes, and 66.7 seconds. There are a couple of problems with this format. First, the longitude and latitude numbers are not base 10 numbers, so it's difficult for an application to determine relative distances. Second, the KVM does not support floating or double primitive types nor the respective wrappers, making any decimal value an incorrect numerical format.

The answer is to use fixed-point integer calculations to handle decimal values. For example, the value of 100.234 is 100234, with a fixed point of three. The GpsParserImpl class instance normalizes the longitude and latitude coordinates by using the following code:

```
int nc = (10000 * k[0]) + (10000 * k[1])/60 +
 (1000 * k[2])/3600;
```

where k[0] denotes the degrees (37), k[1] the minutes (53), and k[2] the seconds (667). The normalized integer value for this latitude is 379019.

To find the speed and direction of the target, we extract information from $GPRMC (fields 8 and 9, respectively). If the altitude field is blank, it means the GPS receiver isn't able to pick up the signal from a fourth satellite, which is necessary for 3D tracking.

In this scenario the target is not moving, so the speed and direction fields contain 0.0. The J2ME application developer may use this information in interesting ways. For instance, you could display all targets moving over 60 m.p.h. in dark blue, or all targets moving north in green. This type of color-coding and dimensional reduction of information allows an enormous amount of location information to be displayed to the user of a limited mobile device. This will undoubtedly be an important area of future growth, research, and standardization for the mobile industry.

Accessing GPS Data Through the Serial Port

Opening the InputStream

Plugging the GPS receiver into the laptop or mobile device starts a stream of GPS data flowing over the serial port. Reading the input stream from the serial port is no different than reading an input stream from a file or an HTTP connection. As mentioned previously, you can use either HttpReader, which functions as a GPS data stream, or SerialReader to open a connection to an actual GPS receiver. Here's the only difference in the code:

- **SerialReader:**

```
    String URI = "comm:1;baudrate=4800;bitsper
  char=8";
    InputConnection inputCon = (InputConnection)
  Connector.open(URI);
    InputStream is = inputCon.openInputStream();
```

- **HttpReader:**

```
    String URI = "http://xspsoft:8080/GPS.html";
    HttpConnection inputCon = (HttpConnection)
 Connector.open(URI);
    InputStream is = inputCon.openInputStream();
```

Note that a SerialReader object opens a connection to the stream on communication port 1; however, your port number may be different. If you choose the wrong port, the KVM will throw an IOException stating that the system can't find the specified file. If the port is occupied, the IOException states that the handle is invalid.

Each GPS receiver sends data at a certain baud rate. Ensure that the baud rate in the String URI matches the baud rate of your GPS receiver. If any of the bytes in the input stream are above 128, the baud rate is incorrect. The input stream will come through as garbage. Also make sure there are no spaces between the semicolons or the program will throw an exception.

Parsing the InputStream

The abstract class GpsParser contains the base code to read the GPS data stream. A concrete, direct subclass instance instantiates either the SerialReader or HttpReader, depending on the user's preference. GpsParser returns an NMEA sentence as a character array, one line at a time, through the use of the Template Method and an abstract method denoted as com-mand(char[] c).

The GpsParserImpl class contains an example of a concrete implementation of the command instance method, which is executed for each NMEA sentence flowing through the serial port. In the GpsParserImpl class, the match-

infragistics, inc.

www.infragistics.com

Command class method determines whether the current NMEA sentence is the GPGGA sentence. If it is, the command method scans and tokenizes the data into a vector.

Invoking the GpsParserImpl constructor causes the GPS data stream to continually update the Coordinate class with the respective longitude and latitude information. Now any application can invoke the Coordinate accessor class methods – getLatitude, getLongitude, getAltitude, getSpeed, and getDirection – to get the most current position of the target.

Data Access Objects and the Transporting of Information

In this article various data sources, including a relational database, XML files, MIDP record stores, and GPS data streams, are coming over a serial port. Dealing with so many data sources can be overwhelming without a systematic design and structure of the application services. Using data access objects (DAOs) to access these data sources is a good design practice that will simplify the application. One example we've already discussed is the coordinate data access object (actually a class) that returns the most current location information of the target.

This application has three primary data access objects on the MIDP device that can marshal and unmarshal XML: SerialDataObject, EntityDataObject, and RecordDataObjectImpl. The SerialDataObject handles the unidirectional transfer of information from the serial port (through Coordinate DAO) to the server database. The EntityDataObject handles the unidirectional transfer from the database to class methods of CoordinateTarget on the MIDP device. The RecordDataObjectImpl handles the bidirectional transfer of information between the server database and the MIDP record store. While transferring data to the record store may be slow, it is, however, a vital component to building enterprise applications involving an MIDP device.

These data objects can be returned by invoking the getDataObject (String DAO) class method of the factory DataObjectFactory. IGps is a superinterface of all these classes, so these class instances are assured of having accessor instance methods for the GPS data. The SessionMinibean object will take care of the marshaling and unmarshaling as well as choosing the correct data access object. For example, to transfer GPS data from the serial port to the server database merely requires the following lines of code:

```
  SessionMinibean smb = new
SessionMinibean("SERIAL");
  smb.marshal();
```

The database server will now contain the user's longitude, latitude, altitude, speed, and direction. Another user, with the following lines of code, can retrieve this user's latitude:

```
SessionMinibean smb = new
   SessionMinibean("ENTITY");
 smb.unmarshal();
 int latitude =
    CoordinateTarget.
      getLatitude();
```

Note that the unmarshal method unmarshals the XML message and populates the class methods of CoordinateTarget, so any application has access to the target's location through CoordinateTarget. Class methods are extremely useful for J2ME applications because of the expense of packing, unpacking, and searching for information within the persistent record store.

If you'd like a local record store copy of the data from the database server, use the following lines of code:

```
  SessionMinibean smb = new
SessionMinibean("RECORD");
  smb.unmarshal();
```

You could also invoke the marshal method and update a database on the server.

In short, we have the basis for wireless hot sync capability between the record store on the mobile device and the database server. Thus the user could set his or her GPS and target tracking preferences, upload the information to the database, and then hot sync from any other mobile device. Hot synchronization could also allow multiple people to synchronize their tracking of a single target or group of targets. Although the record store is not used in this article, the functionality exists within the GPS application download.

A Brief Digression on the MIDP Record Store

The J2ME services on the mobile device access persistent data through the Record Management System (RMS) API. In this example, we avoid the overhead of RMS by accessing location-based information through the use of accessor class methods. However, record store access is critical to any enterprise production application. We'll briefly cover some of the basics.

In the mainframe era space was at a premium, so the flat file was packed in binary and other formats. As a result, programmers built data-access components that packed and unpacked flat files. The same design principle applies to the somewhat more limited mobile devices that use byte records in a flat-file RMS.

The application needs packing and unpacking functionality to store the GPS byte information in a record store. The packBytes instance method in the RecordDataObjectImpl (see Listing 1) writes the values from the getter methods into a DataOutputStream and then converts the stream to a byte array. The setRecord instance method invokes the packBytes method and adds the bytes to the record store.

One issue with the MIDP record store is which record to pack and unpack. In container-managed persistence, the container automatically generates a findByPrimaryKey instance method for the EJB, which returns the object given by the primary key. The GPS application's DAO on the mobile device includes a similar instance method denoted as setPrimaryKey. This is an important method because accessing the data by the record ID creates program-data dependence on the storage structure, resulting in a poor design of the record store.

The trick to locating records by the primary key, rather than key index, is to implement the data access object (RecordDataObjectImpl) as a RecordFilter. This allows the use of a RecordEnumeration on the current object to unpack the record bytes for the record that contains the primary key. By invoking the setPrimaryKey method, a RecordEnumeration returns those records (only one) that are true for the match method (see Listing 2). It then unpacks the correct record, giving access to the record information through the accessor methods. This process is invisible to the business object that instantiates RecordDataObjectImpl.

It's a lot of work to look up and access data from a record store. However, notice that the use of the data access object is almost identical to the CMP entity bean in this GPS applicacompuware corp.

www.compuware.com/products/optimalj

tion. In a sense, the application uses a MiniEJB entity bean on the mobile device. For example, consider the code fragments in Listing 3. For both the MIDP and EJB code, we create a RecordDataObject and set the primary key to an integer that contains the value of 3. In both cases we invoke the accessor methods. The only difference is that in MIDP we must invoke setRecord, which is primarily needed for efficiency.

Note that in both cases the business object doesn't need to know the record ID on which it is operating. By using a similar design for the MIDP and EJB process, the code is considerably reduced in the business object layer.

- **MIDP**

```
 com.xspsoft.j2me.db.GpsDataObject gdo = new
GpsDataObject();
 gdo.setPrimaryKey(new Integer(3));
 gdo.setId(new Integer(10));
 gdo.setName("Mr. X");
 gdo.setRecord();
```

- **EJB**

```
 com.xspsoft.gps.bean.GpsDataObject gdo = new
GpsDataObject();
 gdo.getEntityBeanByKey(new Integer(3));
 gdo.setId(new Integer(10));
 gdo.setName("Mr. X");
```

XML and Data Binding in J2ME

It's worth looking into how the SessionMinibean marshals and unmarshals XML data for the transfer of tabular information. Note that the following method for dealing with XML will also be useful for the serialization of Java objects. The lack of MIDP support for RMI will make XML (and SOAP) a critical component for enterprise applications involving J2ME. Therefore, it's a good idea for the J2ME developer to become proficient in using and manipulating XML.

The SessionMinibean class instance handles the transfer of GPS information between one of the data objects returned by DataObjectFactory and the RootXml and ClientXml objects, which are discussed later. First, we'll demonstrate how the kXML package from Enhydra can read and write XML from the mobile device to a servlet. To write the <client myName="Mr. X" myLatitude="320854"/> tag to the SOAP servlet requires the following lines of code:

```
String URI = "http://xspsoft:
  8080/xspsoft/SOAP";
HttpConnection ic =
   (HttpConnection)
 Connector.open(URI);
 OutputStream os =
    ic.openOutputStream();
 OutputStreamWriter
   writer = new
 OutputStream-
   Writer(os);

org.kxml.io.XmlWriter w =
  new XmlWriter(
    (Writer) writer);
w.startTag("client");
w.attribute("myName",
 "Mr. X");
w.attribute("myLatitude",
 "320854");
w.endTag();
ic.close();
```

The XmlWriter instance requires a Writer parameter, which can be obtained by wrapping the OutputStream with an OutputStreamWriter. Now, writing to the XmlWriter instance will write directly to the servlet.

Reading from the SOAP servlet is very similar. Consider the following code:

```
  InputStream is = ic.openInputStream();
  InputStreamReader reader = new
InputStreamReader(is);

  XmlReader xr = new XmlReader(reader);
  String myName = xr.getValue("myName");
  String myLatitude =
xr.getValue("myLatitude");
  ic.close();
```

Two classes, ClientXml and RootXml, handle the primary work of marshaling and unmarshaling the data. ClientXml has accessor instance methods to set and get GPS information for the class instance. Consider the marshal method from the ClientXml class:

```
public void marshal(XmlWriter writer) throws
Exception {
   XmlWriter w = writer;
   w.startTag("client");
   w.attribute("id", this.getId().toString());
   w.attribute("name", this.name);
   w.attribute("latitude",
 this.getLatitude().toString());
   w.attribute("longitude",
 this.getLongitude().toString());
   w.attribute("altitude",
 this.getAltitude().toString());
   w.attribute("speed",
 this.getSpeed().toString());
   w.attribute("direction",
 this.getDirection().toString());
   w.endTag();
 }
```

To write an XML document to the servlet, use the following code:

```
ClientXml cx = new ClientXml();
cx.setName("Mr. X");
cx.setLatitude(new Integer(327890));
<<more set methods>>
cx.setDirection(new Integer(11));
cx.marshal(anXmlWriter);
```

The output looks like:

```
<client name="Mr. X" latitude="327890" ….direction="11"/>
```

The application can marshal any data access object that implements the IGps interface, which is a two-step process. First, invoke the set methods of a ClientXml instance, passing the return values from the DAO get methods as parameters. Second, invoke the ClientXml marshaling method. The unmarshal instance method is similar and can be viewed in the downloaded code. The purpose of the RootXml class instance is to enumerate through a vector of ClientXml objects, invoking its marshal or unmarshal method. In this specific case the application automatically writes (or reads) a complete XML document that contains multiple rows of

spiritsoft

www.spiritsoft.net/climber

clients and their respective locations.

The general case is more interesting. We now have a powerful technique: any object with accessor methods can marshal and unmarshal XML data without ever directly parsing XML data.

Plotting Target Location

MainDriver is the core class that runs the MIDlet application. This section gives a brief overview of the process and shows how all the previous programs fit together. The application starts with the following lines of code:

```
    public void startApp() throws
    MIDletStateChangeException {
          new GpsParserImpl();
          GpsPlot gp = new GpsPlot();
          display.setCurrent(gp);
    }
```

Invoking the GpsParserImpl constructor begins a thread that starts updating the Coordinate class accessor methods from the GPS receiver. Next we begin another thread by instantiating the GpsPlot inner class. In its constructor, this GpsPlot object starts a TimerTask thread called MiniServlet:

```
public class MiniServlet extends TimerTask {
    public MiniServlet() {
        Timer t = new Timer();
        t.scheduleAtFixedRate(this, 1000, 10000);
    }

    public void run() {
        SessionMinibean serial = new
      SessionMinibean("SERIAL");
        serial.marshal();
        SessionMinibean entity = new
      SessionMinibean("ENTITY");
        entity.unmarshal();
    }
```

MiniServlet schedules its run method to invoke every 10,000 milliseconds. The run method marshals XML data from the Coordinate class to the servlet by invoking serial.marshal(). The servlet then updates the database on the server through the use of JAXB and container-managed persistence.

The run method from MiniServlet then invokes entity.marshal(). This unmarshals XML data from the server database (through the same servlet), updating the CoordinateTarget class accessor methods. In short, we've passed our coordinates to the server database and retrieved the target's coordinates. The target is going through the same process, passing its information to the server and retrieving our GPS information.

The GpsPlot object run method stays alive indefinitely with the following code:

```
while (true) {
   try {
     Thread.sleep
      (100);
     repaint();
   }
     catch
     (Exception e){}
}
```

The repaint method invokes the drawMan method, given below:

```
1.  private void drawMan(Graphics g) {
2.   int lt = Coordinate.getLatitude() - CENTER_Y +
    CENTER_SCREEN;
3.   int ln = Coordinate.getLongitude() - CENTER_X +
    CENTER_SCREEN;
4.   int speed = Coordinate.getSpeed();
5.
6.   g.fillRect(ln - 2, lt - 4, 4, 4);
7.   g.fillRect(ln - 3, lt - 8, 6, 4);
8.   g.fillRect(ln - 2, lt - 9, 4, 1);
9.   g.fillRect(ln - 1, lt - 10, 2, 1);
10.  g.fillRect(ln - 2, lt - 11, 4, 1);
11.  g.fillRect(ln - 1, lt - 12, 2, 1);
12.  g.fillRect(ln, lt, 2, 2);
13.
14.  int deltaLat=Coordinate.getLatitude()-
    CoordinateTarget.getLatitude();
15.  int deltaLong=Coordinate.getLongitude()-
    CordinateTarget.getLongitude();
16.  int distance=com.xspsoft.j2me.util.Math.dist
      (deltaLat,deltaLong);
17.
18.  g.drawString(new
    String("Speed:"+speed),5,76,Graphics.TOP|
      Graphics.LEFT);
19.  g.drawString(new String("Target Dist: " +
      distance), 5, 86, 16|4);
20. }
```

Before the plot is started, the program finds the first latitude and assigns the value to CENTER_Y and the value of the first longitude to CENTER_X. The center of the screen is coordinate pair (50, 50), although this will change depending on the device. On line 2, we calculate the latitude position by invoking the getLatitude method to find the current latitude. Next we subtract the initial latitude. This gives us the latitude movement, centered at zero. We add 50 to center the plot to the middle of the screen. A similar calculation is used for longitude. The fillRect methods plot the pixels that represent a man-shaped figure. As we move, the figure will move from the center coordinate.

We'd also like to display our current speed (lines 4 and 18) and the distance to another target (lines 14–16, 19). Since distance measurements involve square roots, a function not directly supported by the KVM, the download contains a Math class file that handles the distance calculations.

Conclusion

J2ME, J2EE, and XML are helping to end the age of mundane location-based services in the commercial area. The primary advantage is that these technologies open up the location-based services to a larger, more talented pool of developers. Gone are the days when we used our GPS system to ask, "Where is the BurgerBoy exit?" Instead, parents will be tracking their child online or you'll be locating that elusive friend. As a consequence, your elusive friend will be putting a security perimeter around himself to detect other targets so he can remain elusive.

This article demonstrates a simple application using everyday technologies that allows a user to determine his or her distance from a moving object. Its potential is limitless.

random7@worldnet.att.net

AUTHOR BIO
Shane Isbell is a founder and lead developer at xspsoft, which specializes in developng wireless, location-based software for the security industry.

altaworks
corporation

www.altaworks.com

JIM MILBERY JDJ LABS EDITOR

Is the Graphical IDE a Good Thing?

Welcome to the first installment of ***JDJ* Labs**. Our goal is to introduce you to commercial (and open-source) products and technologies that will help you, the Java developer, work more efficiently. We expect our testing to provide you with a starting point for your own testing and analysis. In this column we'll be looking at common market trends that affect the commercial products and services in the Java space.

Over the past few years we here at *JDJ* have looked at quite a lot of Interactive Development Environments (IDEs) for Java programming. IDEs provide a sort of one-stop shop for software authoring, compiling, and deployment. Historically speaking, software programs used to be written using a regular OLE text editor. These original editors treated text documents and code the same way – as text. Specialized text editors assisted programmers with the job of writing and editing code.

Even many popular 4G languages make use of a text editor as the primary programmer's interface. It was the emergence of graphical user interfaces (especially Windows) that fomented the emergence of the full-blown IDE. This new breed of IDE handled all aspects of programming – development, coding, debugging, testing, and deployment. Over the years the graphical IDE has become a de facto standard of sorts. In fact, all the leading tools and database vendors were quick to release their own graphical Java IDEs almost as soon as the Java revolution began.

The question that faces us, however, is whether or not the graphical IDE is a good thing. On the one hand it provides a one-stop shop for programmers to gain access to all the necessary parts of a programming language. On the other, graphical IDEs tend to insulate the programmer from having to understand all the ins and outs of a particular language. Personally, I believe it's important for a developer to have a firm understanding of the underlying language (especially when that language is more of a platform – like Java).

I don't mean to imply that a programmer needs to have an intimate understanding of the internals of each and every class that they use. (After all, one of the premises of OO development is the ability to rely on the published interfaces of an object.)

The problem, to my way of thinking, is the degree to which most IDEs insulate developers from their code. For example, graphical IDEs are filled with wizards that take care of the common tasks a programmer faces. You want to build a Swing-based data entry form for a table in a database? Just use the form-builder wizard. The problem with this approach is that the programmer may never understand the logic used to generate this form. Generated forms rarely meet 100% of your needs, and the novice developer winds up trying to modify a complex set of Java code that was created by the wizard.

Wouldn't it be a better long-term approach for this same developer to build his or her first few data entry forms from scratch and learn the ins and outs of Swing and JDBC? I've also heard experienced developers weigh in with the argument that most IDEs just "get in the way" when it comes to Java coding. To be sure, some of this comes from the age-old adage that "real programmers use vi or Emacs, or <insert your favorite text editor here>." However, there is a nugget of truth here as well. My question to you, dear reader, is where you stand on this issue. To IDE or not to IDE – that is the question!

• • •

Send your comments to jdjlabs@sys-con.com.

jmilbery@kuromaku.com

AUTHOR BIO

Jim Milbery is a software consultant with Kuromaku Partners LLC, based in Easton, Pennsylvania. He has over 17 years of experience in application development and relational databases. He is the author of Making the Technical Sale.

LABS INDEX

Is the Graphical IDE a Good Thing?
Welcome to the first installment of JDJ Labs. Our goal is to introduce you to commercial (and open-source) products and technologies that will help you, the Java developer, work more efficiently.
by Jim Milbery
128

FrontierSuite 2.2.02
by ObjectFrontier, Inc.
FrontierSuite is a single tool that can help you design, develop, and deploy your e-business Java applications from start to finish.
reviewed by Ramesh Chandak
130

eXtend Workbench
by SilverStream
This application ties together all the tools you would need to create a J2EE/Web services application and puts them into one easy-to-use environment.
reviewed by Jeff Ford
134

Oracle9*i* JDeveloper
by Oracle Corp.
Oracle9*i* JDeveloper allows developers to take advantage of the power and flexibility of J2EE to create applications with the highest level of scalability and performance.
reviewed by John Walker
140

Java Internationalization
by Andrew Deitsch, David Czarnecki, and Andy Deitsch
Java Internationalization is a good resource for developers trying to design code that supports globalization. Besides being well organized, to my knowledge it's the only book that gives undivided attention to this topic.
reviewed by Ajit Sagar
146

ilog

www.ilog.com/jdj/jviews

FrontierSuite 2.2.02

by ObjectFrontier, Inc.

REVIEWED BY RAMESH CHANDAK *chandakrus@yahoo.com*

info

ObjectFrontier, Inc.
2050 Marconi Drive
Suite 300
Alpharetta, GA 30005
Phone: 770 777-8180
Web: www.objectfrontier.com
E-mail: sales@objectfrontier.com

Test Environment
Processor: 966MHz Intel Pentium III processor
Hard Drive: 20GB Disk
Memory: 256MB RAM
Platform: Windows 2000 Server and Oracle Enterprise Server 8.1.7i (optional)

Specifications
Platforms: Based on the Java platform and compliant with model-driven architecture
Pricing: A 30-day evaluation copy is available for free. For product-specific pricing, contact the company.

FrontierSuite is a single tool that can help you design, develop, and deploy your e-business Java applications from start to finish. You can start your project with UML-based object modeling using Frontier Modeler and move on to generate EJB entities and deploy using Frontier Deployer. ePersistJ serves as CMP (container-managed persistence) and BMP (bean-managed persistence).

FrontierSuite

Providing a persistence framework for Java is extremely challenging. FrontierSuite meets this challenge head-on with its set of integrated tools. It offers a simple interface to provide the persistence framework, along with a powerful business-modeling tool and an efficient runtime environment for building enterprise Java applications. It's independent of any database – relational or object-oriented – but any implementation for a specific database can be plugged into it.

The product is targeted at enterprise Java developers, including technical architects, business analysts, and Java developers. FrontierSuite is composed of:

- ***Frontier Modeler:*** A powerful tool for creating enterprise object models using UML standards
- ***Frontier Builder:*** A code generator that helps generate enterprise classes, such as EJBs
- ***Frontier Fusion:*** A tool that maps objects, including relational databases and/or ERP (Enterprise Resource Planning) to your EIS (Enterprise Information System)
- ***Frontier Deployer:*** A versatile tool that allows you to generate portable deployment descriptors, such as WebLogic- or JBoss-specific configuration descriptors, along with an application server
- ***EPersistJ:*** An EJB 2.0-compliant unified bean persistence manager, it's configurable to serve as a container-managed persistence manager for EJB servers and a bean-managed persistence manager for EJB applications. Built on JCA architecture and capable of persisting in a relational database, EPersistJ provides distributed caching to enhance the application's performance and synchronize objects through the object notification mechanism.
- ***Frontier ReModeler:*** Provides a mechanism to read an existing schema from a relational database and an environment to map and generate entity beans and their relations.
- ***Frontier DeployDirect:*** Provides CMP for "deployable JAR" files created using third-party tools (EJB 1.1 and 2.0).

Installing FrontierSuite

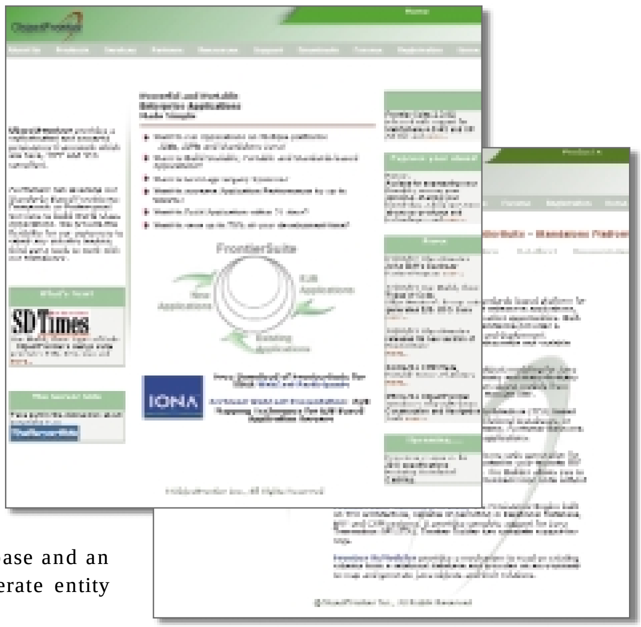
You can download a 30-day evaluation version of FrontierSuite from ObjectFrontier's Web site. Since the product is based on Java, you can install it on Windows, Unix, Linux, or Solaris platforms.

Although it isn't difficult to install, ObjectFrontier assumes you've installed Java 2 SDK, Enterprise Edition. It also requires you to have JDBC libraries on your machine. The installation documentation doesn't provide any guidance regarding which version of SDK is required. I used Java 2 SDK, Standard Edition, version 1.3.1_01; Java 2 SDK, Enterprise Edition, version 1.3; and Oracle JDBC libraries.

The installation wizard guides you through the installation process, during which it asks you for the license key. This is stored within the KeyValue.txt file under the LicenseKey directory. *Note:* This license is valid for 30 days. The remaining installation is pretty straightforward and takes less than five minutes.

I began my project by developing the object model using Frontier Modeler (FrontierSuite.bat), as shown in Figure 1. Frontier Modeler offers a very intuitive user interface. The left pane provides a tree view of the Java classes used and their relationships to each other; the right provides a description of the object highlighted in the left pane. You can use the right pane to add to or to modify the object's attributes. To delete any object, highlight it in the left pane and right-click. A pop-up menu will appear, allowing you to select the delete option.

The Frontier Modeler toolbar allows you to move between model entities and edit their properties. The Enterprise toolbar provides you with the Standalone Platform and J2EE Builder



loox

software inc.

www.loox.com

properties and displays Frontier Builder. The Fusion toolbar allows you to set the relational properties and display Frontier Fusion for different relational databases, such as Oracle, Microsoft SQL Server, Sybase, DB2, Informix, Cloudscape, Microsoft Access, and PointBase.

If you've already developed a model using Rational Rose (www.rational.com) or Together ControlCenter (www.togethersoft.com), you can import it into Frontier Modeler.

Next, I selected the Oracle Fusion submenu option from the Frontier Fusion menu (you can also use the toolbar) to generate a default mapping of objects and their relationships, and create database schema and table entities for the Oracle database. You can also generate them for Microsoft SQL Server, DB2, Cloudscape, and PointBase databases.

Similarly, using the Frontier Builder option (you can also use the toolbar), I could identify the EJB, dependent objects, and dependent value objects for my object model. Frontier Builder generates abstract schema code in addition to defining the home and remote client interfaces for the EJBs. Further, I could add business-rule methods to the entity beans and dependent objects. You can use Forte or JBuilder to develop your business rules and import them into Frontier Builder.

One of the nice things about FrontierSuite is that it includes Frontier Deployer (DeployDirect.bat). Within Frontier Deployer, select the application server platform (you can select WebLogic 6.1 or JBoss 2.4.x) and generate the deployment JAR files. You're now ready to deploy the deployment JAR file within the application server.

Summary

ObjectFrontier impressed me with their niche product. The ability to quickly design and develop e-business applications is simply great. Since the tool generates EJB entities right from the model, development time is significantly reduced. All in all, I feel that FrontierSuite is a great tool for enterprise Java application development.

Editor's Note: Since this review was written a new version of FrontierSuite has been released, version 2.3.02.

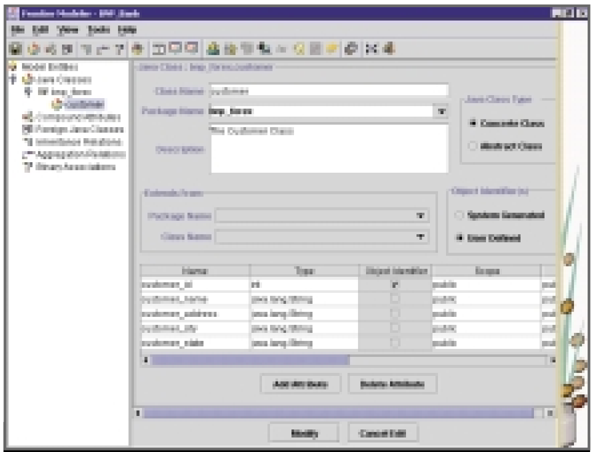


FIGURE 1 Frontier Modeler with the customer class

Product Snapshot

Target Audience: Technical architects, business analysts, Java programmers

Level: Intermediate to advanced

Pros:
- Feature-rich product
- Provides complete development environment from the object model to deploying the classes
- Useful examples
- Applications built using FrontierSuite are portable across application servers

Cons:
- None

JAVA DEVELOPER'S JOURNAL

Next Month

Pervasive Computing: The Next Generation of Consumer Applications

Now that broadband connections are becoming commonplace and wireless technology is becoming a reality, developers should begin positioning their applications to leverage the new capabilities.

by Dan Pilone

Product Review: CAST JavaMiner

JavaMiner may be the answer for people who need to communicate a working application when time is limited.

reviewed by Jason Bell

Java Jobs: The Next Big Thing

From grass roots developer communities to the major industry giants, the message is becoming very clear: get on board for Web services.

by Bill Baloglu and Billy Palmieri

JDiff — What Really Changed?

One of the most common questions Java developers ask after downloading a new version of a product is: "What really changed?" JDiff is an open source Java tool, based on Javadoc and developed by the author, that produces HTML documentation describing the precise API changes between two versions of a product.

by Matthew B. Doar

J2EE Application, Module and Component Packaging

The different types of J2EE modules and how they fit into the J2EE architecture.

by John O'Shea

reportmill software, inc.

www.reportmill.com

eXtend Workbench

by SilverStream

REVIEWED BY JEFF FORD jeff.ford@catavo.com

As the Internet continues to grow as a viable medium for enterprise-class applications, the tools and technology for developing these applications continue to advance at a frenetic pace. The state of the technology now allows for a multitiered application that involves anything from simple scripting to complex objects. Such technological advances have created an environment in which multiple developers can work together on a single "logical" application.

The J2EE specification defines a development model and provides supporting standards and tools for developing multitiered Web-based applications. New standards are being defined in the Web services arena that extend applications across enterprise boundaries. Vendors are working feverishly to harness their integrated development environments (IDE) to include the necessary tools and features to address the needs of Java-centric Web services programmers. SilverStream eXtend Workbench 1.0 (XWB) is a new breed of software that has been designed to address this burgeoning market.

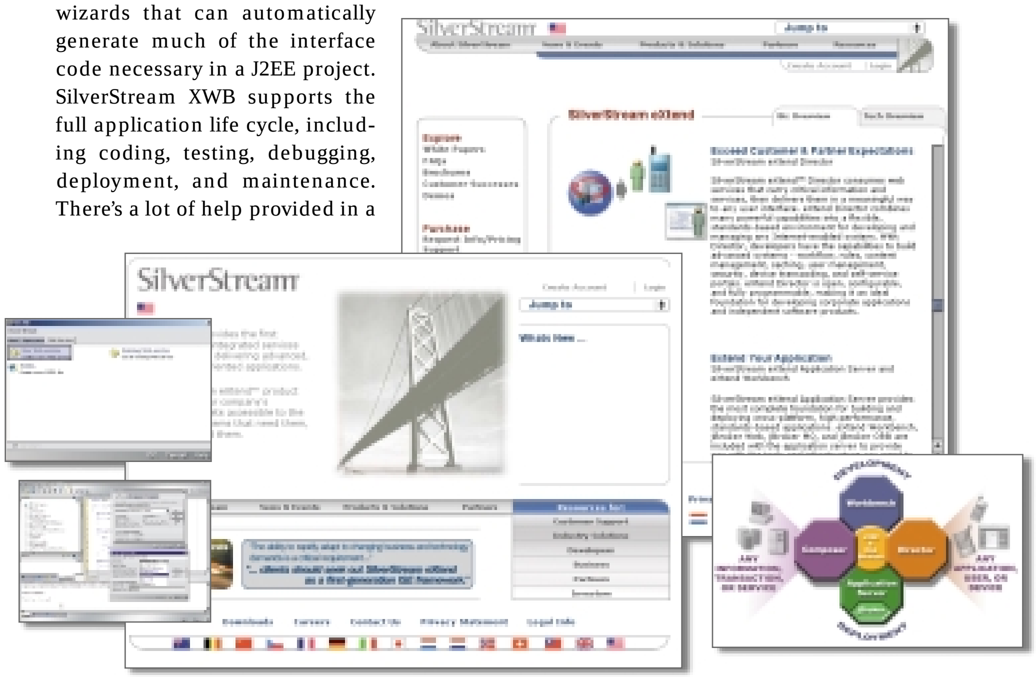
Overview

I like the name *workbench*, because it sums up what this product is all about. This application ties together all the tools you would need to create a J2EE/Web services application and puts them into one easy-to-use environment. It follows the J2EE development model and is capable of building standards-based applications for any J2EE-compliant server. It speeds the development cycle by providing wizards that can automatically generate much of the interface code necessary in a J2EE project. SilverStream XWB supports the full application life cycle, including coding, testing, debugging, deployment, and maintenance. There's a lot of help provided in a set of HTML-based help files, as well as a set of tutorials and sample applications. The eXtend Workbench Web site (www.silverstream.com/workbench) also has downloads, discussion forums, news, tips, and techniques.

The eXtend Workbench comes with the SilverStream eXtend Application Server that's fully J2EE 1.3–compliant and supports many of the latest Java technology and Web services standards including EJB 2.0, SOAP 1.1, and JMS 1.0.2. The application server was one of the first to pass all necessary certification tests and SilverStream is dedicated to supporting future versions of J2EE as they are approved. Still, the application server is a separate installation, and Workbench does integrate with most of the major third-party application servers on the market, including BEA's WebLogic (6.0 and higher) and IBM's WebSphere 4.0. (SilverStream includes ample documentation and samples that detail how to integrate XWB with other application servers.)

Working with the Product

Installing the XWB was quick and easy. However, the eXtend Application Server took a little longer to install, and it expects to have a database already running on the server. I happened to have Microsoft SQL Server already installed, so I configured the eXtend Server for that database, which turned out to be pretty simple. The XWB application has a rich graphical look to it. I've come to expect that most Java IDE tools today are built using Java Swing code , so I was not surprised to see a "Java" look to this application. I did find it interesting that this version of the eXtend software was targeted specifically toward Windows-based platforms.



info

SilverStream Software
2 Federal St.
Billerica, MA 01821
Web: www.silverstream.com
Phone: 888 823-9700
E-mail: info@silverstream.com

Test Environment
Computer: Gateway Solo 9300
Processor: Pentium III 333 MHz
Memory: 160MB RAM
Platform: Windows NT Workstation 4.0 (Service Pack 4)

Specifications
Platforms: Windows NT Workstation 4.0, Windows NT Server 4.0, or Windows 2000
Pricing: Free for a limited time to J2EE and Web services developers. Application Server Developer Edition includes a single version of the application server and the eXtend Workbench development environment, priced at $495 per developer seat. Workgroup Edition (Windows and Unix) limited to 25 connected users, priced at $995 per CPU. Enterprise deployment pricing begins at $5,000 per CPU.

simplex
knowledge
company

skc.com

some of the best things in life are free
JAVA DEVELOPER'S JOURNAL
FREE CD Inside
PDAs
Getting Started with Java
Download HPAS 8.0 FREE

XWB's edit layout, shown in Figure 1, provides an environment that seemed familiar and well laid out for me. There are tree and list views on the left to navigate through project files and directories, an editor window on the right, and an output window on the bottom for informational messages. The window panes are resizable, and it has a toolbar, popup menus, and keyboard shortcuts like any good IDE. The code editor had most of the useful features I rely on in my favorite text editor, including autoindentation, autocompletion, and syntax highlighting.

For an even richer coding environment I would have liked bookmarks, the ability to split the code screen into two sections, and a tabbed view of all open files. (There didn't seem to be any flexibility to rearrange the various screen elements.) The Java code compiler gives the usual Java error messages on compilation, but the environment has an extremely useful feature – if you click on an error message, the cursor will jump to the offending line of code in the editor pane.

The wizards are very useful in providing enough options while not being overly complicated. Another useful feature is the flexibility in project starting points. From the XWB you can start

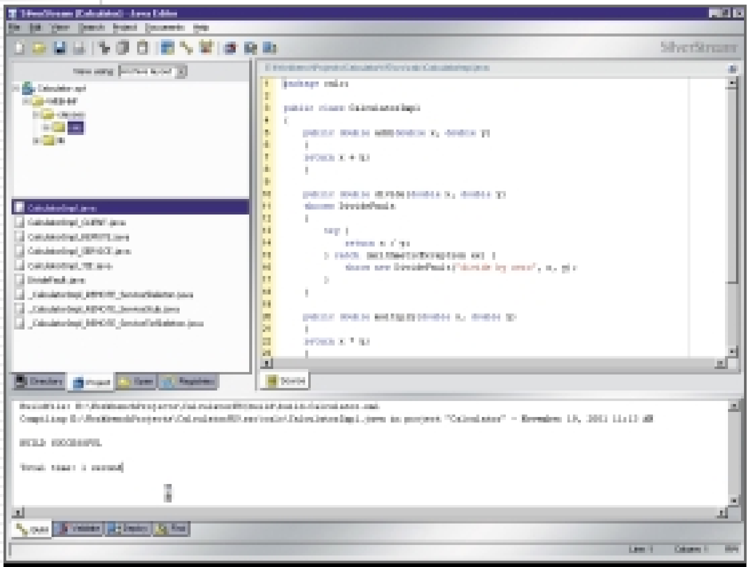


FIGURE 1 eXtend Workbench environment

a project from scratch or from an existing J2EE project. Once you create a XWB project, you can import existing files into the project. This way, you can start with a class that implements business logic functions, and build that class into a J2EE Web service with just a few clicks through the Web service wizard. Or, you can start from an interface definition and build your project up from there (see Figure 2). One powerful and convenient function is XWB's ability to archive and deploy J2EE projects to any of the supported J2EE application servers.

Either way, the XWB Web service wizards and editors allow the developer to focus more on the business logic, since much of the Web service framework is generated for them. SilverStream also included a comprehensive Registry Manager tool that can help you browse or search for existing Web services, or publish your own using Universal Description, Discovery, and Integration (UDDI). This is a useful feature for developers trying to integrate with other Web services, since they can import WSDL definitions without leaving the XWB environment.

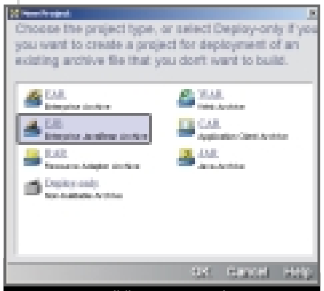
While I only tested this with the SilverStream eXtend Application Server, the configuration process was straightforward, and once configured, deployment was as easy as clicking a

jdjedge
conference
& expo
www.sys-con.com

button. I believe that's one of the great advantages of environments such as SilverStream's eXtend. Once configured, tasks that would involve long and complex command-line program deployment calls can be simplified into one click of the mouse. This allows developers to focus on design and coding issues, and worry less about which parameters indicate deploying only changed files.

As for speed, the application seemed to respond well on my test platform, as long as I didn't try to run too many other applications simultaneously. I'm the type of person who likes to have five to six programs running at a time, and I began to notice some lag in switching back and forth. I wouldn't really be concerned, however, because with faster hardware and more memory, I'm sure that this level of multitasking would perform just fine. Compiling, archiving, and deploying projects from within the environment all seemed to run well.

SilverStream's eXtend Workbench helps speed up the development of J2EE projects and Web services applications. I'd recommend that the developer have some basic knowledge of the various components in a J2EE project and Web services standards before using a comprehensive product such as eXtend. The tutorials do a wonderful job of walking you through the implementation process. However, a reasonable background in Java and Web services will make it much easier to harness the power in SilverStream's eXtend product line.


New Project

Choose the project type, or select Deploy-only if you want to create a project for deployment of an existing archive file that you don't want to build.

FIGURE 2 Building a new project

Summary

SilverStream eXtend Workbench does an excellent job of providing all the tools you'll need to develop complex J2EE/Web services projects using a single IDE. Integration with the leading application servers makes eXtend an ideal development environment if you already have an application server – or you can use the SilverStream eXtend Application Server. Java developers with little or no knowledge of J2EE design concepts may have trouble being efficient with this tool at first, but with a basic understanding of J2EE, I believe XWB will help any developer be more efficient in designing, developing, and deploying a J2EE/Web services application.

Product Snapshot

Target Audience: Java J2EE developers

Level: Entry-level J2EE (with some background knowledge) to advanced

Pros: Open and extensible IDE integrates with other tools and works with several industry-standard application servers; additional product offerings are available.

Cons: As with many Java-based applications, you'll want a powerful machine to get good performance. Not for Java beginners. eXtend Workbench only supports Windows-based platforms.

REGISTER NOW FOR THE 2002 WORLD TOUR!
XML JOURNAL
Development Tools for XML Applications

xmledge
conference
& expo

www.sys-con.com

Oracle9*i* JDeveloper

by Oracle Corp.

REVIEWED BY JOHN WALKER *walk@sbcglobal.net*

Oracle Corporation
500 Oracle Parkway
Redwood Shores, CA 94065
Web: www.oracle.com
Phone: 650 506-7000

Test Environment
Computer: Dell Precision 340 Workstation
Processor: 1.80GHz Intel Pentium IV processor
Hard Drive: 40GB Disk
Memory: 512MB RAM
Platform: Windows 2000/SP 2

Specifications
Platform: Any platform with JDK 1.1 or 1.2 support
Pricing: A free evaluation copy can be dowloaded at http://otn.oracle.com. A full license, costing $5,000 per developer, is required for applications that will be sold or deployed in a production environment.

Anyone who has used the Java 2 Platform, Enterprise Edition (J2EE) has to be impressed with how quickly it's matured and become a robust programming model. Besides the ability to build new applications, developers can use J2EE to connect to third-party software, legacy systems, and Java-based e-business application engines and deploy them across a distributed computing environment. As powerful as the various technologies supporting this platform are, efficiently managing and using them all in a development environment can be problematic. Providing an IDE to manage this complexity is what Oracle has set out to accomplish – and in my opinion, has very successfully – with its new Oracle9*i* JDeveloper product.

Oracle9*i* JDeveloper allows developers to take advantage of the power and flexibility of J2EE to create applications with the highest level of scalability and performance. Written entirely in Java, it's a robust IDE that combines all of the tools and services to comfortably support a developer throughout the full J2EE development life cycle. Oracle9*i* JDeveloper also includes an open J2EE framework called Business Components for Java (BC4J) that helps developers quickly construct high-performance J2EE applications following industry-standard J2EE design patterns.

For database-backed J2EE applications, the framework makes the object-relational mapping process a point-and-click exercise. For client applications it includes an extensive set of JSP tags in a built-in tag library and a large set of Swing-based components. Easy-to-use wizards speed the creation of JSP-, servlet-, EJB-, and Web services-based components. Built-in container objects then allow for the quick and very easy testing of these components without you ever having to leave the development environment. It's conceivable that you won't have to leave the confines of the JDeveloper environment for a wide range of development services, including source control, UML modeling, debugging, testing, profiling, and deployment. Developers can also use the AddIn Kit to integrate third-party code libraries and open-source products, which could include their own custom tag libraries.

Installation and Configuration of JDeveloper

The installation of the product is straightforward and all necessary files are made available in a zip archive format. After the files have been extracted, a shortcut should be created to the file [jdeveloper_root]/jdev/bin/jdevw.exe to execute the application. The necessary documentation is available from the online help, the free product download from the Oracle Technology Network (OTN) at http://otn.oracle.com.

One of Oracle9*i* JDeveloper's most exciting features is the built-in Oracle9*i*AS Containers for Java (OC4J). With OC4J, developers aren't required to have access to a separate application

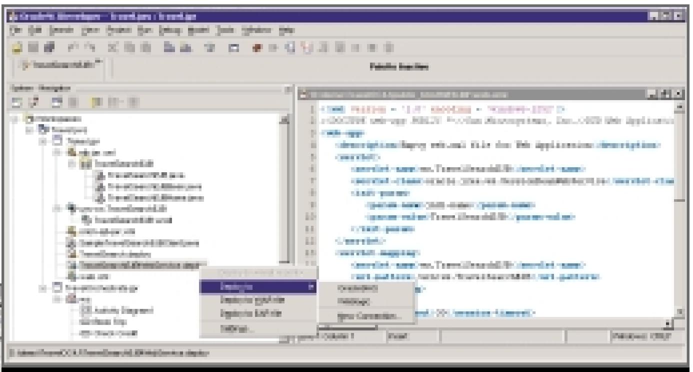


FIGURE 1 Test Deployment

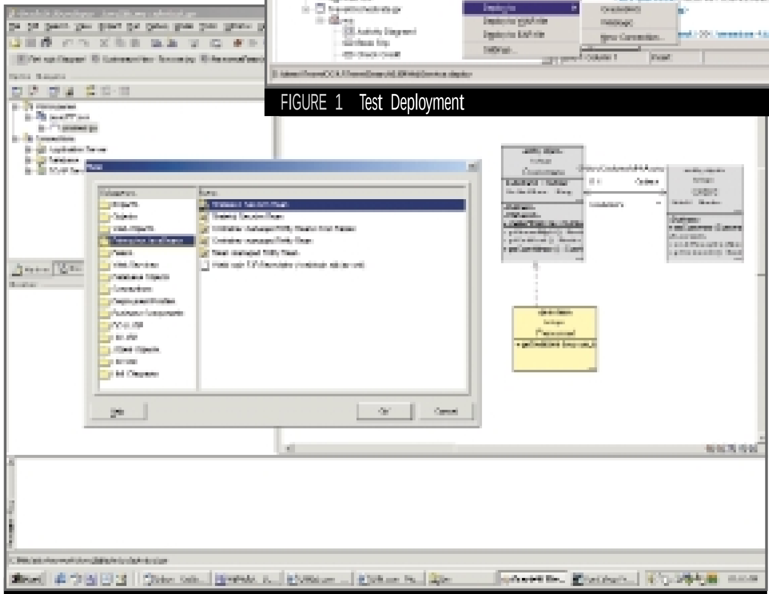


FIGURE 2 JDeveloper's IDE featuring New Gallery and UML Modeler

web services
edge conference
& expo

www.sys-con.com

Oracle9*i* JDeveloper by Oracle Corp.

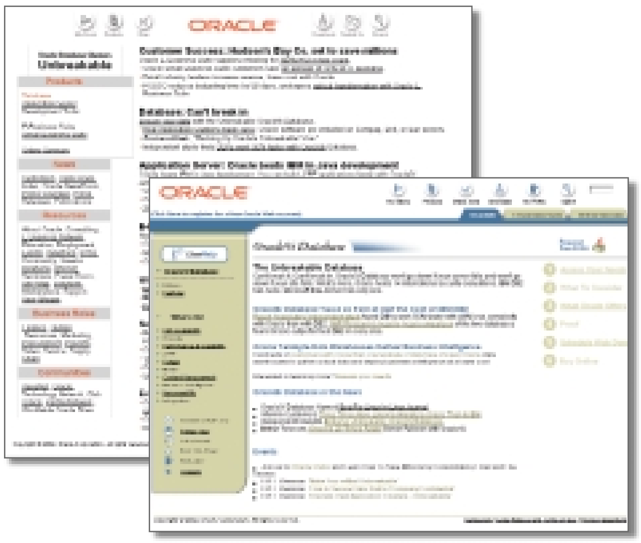
server to test the components they're building within the JDeveloper environment. OC4J is built into JDeveloper, allowing for the immediate testing of JSP, EJB, servlet, and Web services components.

With a little extra work an OC4J server can be run remotely to simulate a preconfigured J2EE deployment environment. The steps to implement this feature are few and well documented in the installation guide. I highly recommend implementing the OC4J remotely. A two-step process, it's the final touch that makes this elegant IDE a completely integrated development environment. The remote implementation allows for the test deployment of EAR, JAR, and WAR files via the various deployment wizards JDeveloper offers (see Figure 1).

The features are easy to use with the sample code and database configuration files supplied from the OTN Web site. Oracle 9*i* JDeveloper works with any JDBC-compliant database, including Oracle Database release 8.1.7 (or higher). After setting up the database, you'll need to create database connections to the various schemas and to the application and SOAP (Simple Object Access Protocol) servers. (If no application or SOAP servers are available, OC4J conveniently takes their place for testing and deployment of J2EE components and Web services.)

Working with Oracle9*i* JDeveloper

From setup to the creation of the first objects, the ease with which the environment allows the developer to accomplish tasks is impressive. The sample code and associated documentation is excellent and provides a quick entrée into the important features of this product and how they work. On my own I created a simple application to investigate the feature sets and see how well they work together. I started out



by creating a simple UML model within JDeveloper that would allow tracking of "customer" orders. From this initial model I was curious to see how easy it would be to implement the business components generated from the model into the various component types (servlets, JSPs, EJBs, etc.) that are supported by the J2EE platform.

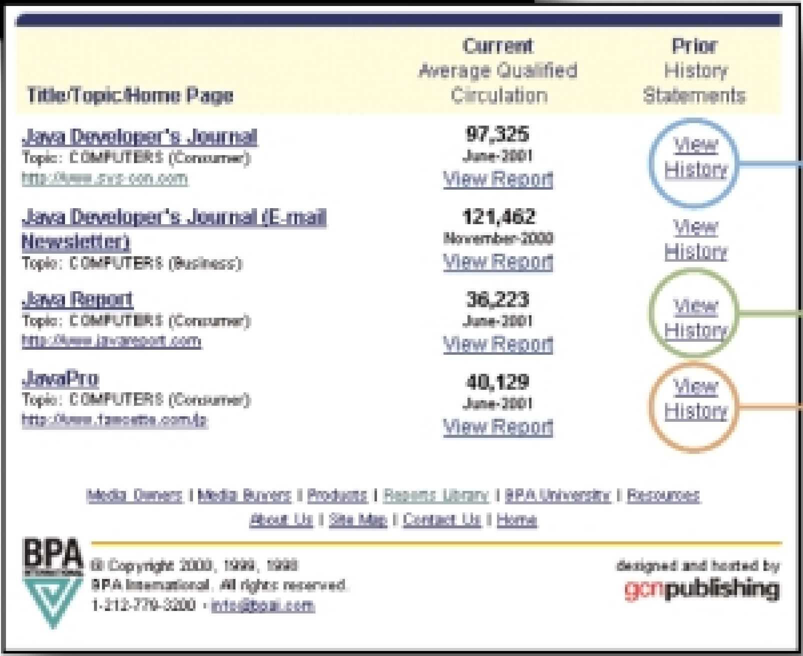
The high degree of integration makes an immediate impression upon entering the Oracle9*i* JDeveloper environment. Developers can have simultaneous connections to database, application, and SOAP servers across multiple workspaces and projects. A common interface, the New Gallery dialog, is used whenever developers need access to a wizard to create an object within the IDE (shown in Figure 2). This interface is used whether it's a new workspace, EJB component, or database connection to be created. This is an elegant interface that reduces the confusion over where to go to find the appropriate wizard to create the desired object or component.

"As the Java print media market consolidates, *Java Developer's Journal* will most likely be the only long-term player..."

– Scott Clark, Editor of *Java Boutique*/MecklerMedia

Now that our competitor is no longer in print...

BPA Web site: October 15, 2001



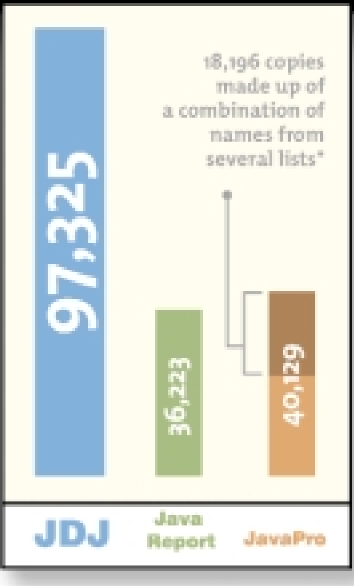
Title/Topic/Home Page	Current Average Qualified Circulation	Prior History Statements
Java Developer's Journal Topic: COMPUTERS (Consumer) http://www.sys-con.com	97,325 June-2001 View Report	View History
Java Developer's Journal (E-mail Newsletter) Topic: COMPUTERS (Business)	121,462 November-2000 View Report	View History
Java Report Topic: COMPUTERS (Consumer) http://www.javareport.com	36,223 June-2001 View Report	View History
JavaPro Topic: COMPUTERS (Consumer) http://www.fawcette.com/jp	40,129 June-2001 View Report	View History

Media Owners | Media Buyers | Products | Reports Library | BPA University | Resources
About Us | Site Map | Contact Us | Home

© Copyright 2000, 1999, 1998 BPA International. All rights reserved. 1-212-779-3200 · info@bpai.com

designed and hosted by gcnpublishing

Current Circulation: June 2001



* see BPA Statement

...*JDJ* is here to serve you as your exclusive media partner for many years to come!

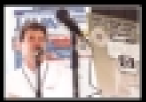
Historical Facts: June 2000

	Rate Base	Actual Circulation
JDJ	60,000	67,618
Java Report	30,000	23,777
JavaPro	30,000	24,179

carmen@sys-con.com

miles@sys-con.com

For accurate information on the circulation of Java publications, and the hands-down leadership position of ***Java Developer's Journal*** and how it relates to your advertisement cost per thousand copies (CPM), please contact your account manager today.



"*Java Developer's Journal* gives us the highest number of leads on any advertising we do..."
Larry Humphries, President of Sitraka LIVE on SYS-CON Radio

SYS-CON MEDIA
www.JavaDevelopersJournal.com
(201) 802-3020

Oracle9*i* JDeveloper by Oracle Corp.

The creation of the UML model was expedited by a previously created connection to an existing database schema containing the table definitions that would form the basis of the UML model. Accessing the entity definitions from the database made the UML creation a simple drag-and-drop exercise. After that it was just a matter of deciding which wizard to use to implement the desired component from the object definitions represented in the diagram. The first step in this process is to generate business components from the diagram into the project. The components represent the objects in the model, and serve as templates for the creation of J2EE components. Attributes and methods can be added before they're rendered into JSP, servlet, or EJB components.

Using the various wizards provided by the New Gallery dialog, the customer and order objects were quickly and efficiently implemented as JSP, entity EJB, and Web services. The integration between the various supporting files of the EJB made a particular impression. Whenever a new method was added to (or changed, for that matter) a bean definition file, there was no need to make changes to the supporting interface files since they were implemented automatically, thus eliminating a big headache all EJB implementers have had to deal with at one time or another. The customer entity was quickly implemented as a container-managed entity bean with the necessary support files, including the deployment descriptor file. Deploying the new EJB to the application server was a point-and-click exercise after creating the deployment JAR file with the appropriate wizard.

For testing purposes, JDeveloper creates a sample Java client to access the EJB definition; the code generated for this client was terrific and I used it later as a template for the implementation of a stateless session bean to manage my customer EJB.

As a final step when implementing some custom code I wrote for the stateless session bean, I ran CodeCoach for guidance concerning its efficiency. CodeCoach provides insights into issues ranging from whether certain segments of code are ever used to whether a variable's scope should be changed for better use of memory resources. During the testing process, memory usage can be tracked using the integrated memory profiler.

Summary

Oracle has provided a quality product that is up to the task of managing the J2EE application life cycle. Easy-to-use wizards are the starting point for creating components that take advantage of the variety of built-in services the J2EE platform provides, while reducing the complexity of implementation. These same wizards offer a final conduit to the application or SOAP servers necessary for deploying components. The integration and ease of use provides an environment that can manage projects, and a technology for that matter, even as they scale – concepts that experienced and beginning developers alike can relate to.

JDJ Product Snapshot

Target Audience: Java programmers, application architects

Level: Beginner to advanced

Pros
- Feature-rich product
- Built-in J2EE container for testing
- Powerful wizards for quick and integrated component creation
- Easy integration of third-party APIs and frameworks

Cons
- None significant

e-BUSINESS SOLUTIONS: TAKING BUSINESS TO THE NET
SPECIAL: PREMIER ISSUE!
WebSphere DEVELOPER'S JOURNAL
INTRODUCTORY CHARTER SUBSCRIPTION SAVE $30 Today!
Iterative Development with VisualAge for Java
Best Practices in WebSphere Application Packaging and Deployment
Iterative Development with VisualAge for Java
Building an Application Server Data Center
IBM WebSphere Studio v3.5 Advanced
It's You, Not the Software

Do You Have Access to the Internet?

Then Subscribe Online and Save $31!

It's that easy

Simply point your mouse to...

WebSphereDevelopersJournal.com

The World's Leading Independent WebSphere Developer Resource

Introductory Charter Subscription

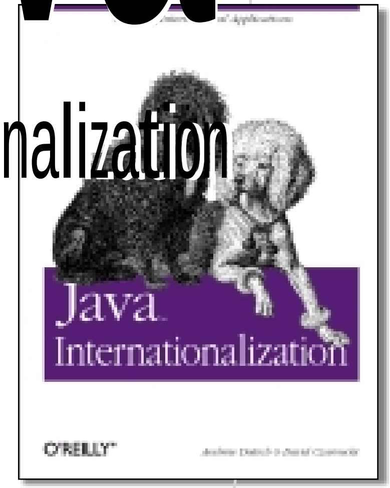
WebSphere DEVELOPER'S JOURNAL

SUBSCRIBE NOW AND SAVE $31.00 OFF THE ANNUAL NEWSSTAND RATE

ONLY $149 FOR 1 YEAR (12 ISSUES) – REGULAR RATE $180

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

Java Internationalization



REVIEWED BY Ajit Sagar ajit@sys-con.com

info

Java Internationalization
Authors: Andrew Deitsch, David Czarnecki, and Andy Deitsch
Publisher: O'Reilly & Associates

As enterprise applications go global, the need for applications that can be customized across different languages and locales is becoming paramount. Java has provided internationalization capabilities since JDK 1.1. However, information on this very crucial topic has been scattered across different tutorials and sites. I was glad to see a book focused purely on internationalization in the Java application space. *Java Internationalization* is a good resource for developers trying to design code that supports globalization. Besides being well organized, to my knowledge it's the only book that gives undivided attention to this topic.

This is a book for programmers. It has good information on the topics of internationalization, localization, and globalization in general. It offers information on designing internationalized code for both Web-based and client/server applications. However, its main focus is on how to write internationalized Java code. If you're looking for a general book on i18n, this may not serve your purposes. The coverage on Java internationalization is quite extensive. There is ample discussion and plenty of examples on Java's built-in support for locale; the nuances of writing code to safeguard data formats for text, dates, currency, different markets, numbers, time zones, etc.; and Java APIs like resource bundles.

The book starts with a general introduction to the concepts behind internationalization. Although the focus is on Java, the first three chapters offer general information that will be useful to folks new to i18n concepts. Chapter 4 covers the usage of resource bundles in Java in detail. This is one of the most useful chapters for Java programmers. Chapters 6–8 focus on specific areas of programming where developers must make sure their code complies with the design guidelines for internationalization. Chapter 7 also provides ample coverage on Unicode.

The remaining chapters focus on internationalization for specific application types. Chapter 9 focuses on GUIs; Chapter 11, on Web-based applications. The coverage is mainly related to the Windows platform. The book concludes with a road map for the future evolution of Java 3.0 internationalization and a reference that covers Java APIs for i18n.

Although this is a good reference for developing internationalized code in Java, if you're putting together an internationalization strategy and standards for your applications, you'll still need other sources of information as well as examples of how it's done in the trenches. This book deals with the subject primarily from the Java language point of view. The examples are lucid and well explained. However, I would have liked a "Tips & Tricks" or a "How-to" section. It would be great if the authors can provide these, perhaps as supplementary chapters in the next edition.

As far as the overall organization of chapters goes, I felt that the second chapter interrupted the flow of the book. Although there is good information, not much of it has to do with how you internationalize your code in Java. This chapter would probably fit better into an appendix.

All in all, *Java Internationalization* is a good book, with the right level of detail and enough illustrations, in the form of code examples, snippets, and screenshots, to be a good reference for when you embark on a project requiring global deployment.

Table of Contents

Preface

1. Introduction to Internationalization
- What Are Software Internationalization, Localization, and Globalization?
- Why Choose Java for International Applications?
- What is a Locale?
- A Simple Application

2. Writing Systems
- Ancient Writing Systems
- Far East Writing Systems
- Bidirectional Scripts
- Greek, Latin, and Cyrillic
- Indic Scripts
- Thai Script
- Punctuation

3. Locales
- Defining a Locale
- Working with the Locale Class
- Querying for Locale Information
- Checking Available Locales

4. Isolating Locale-Specific Data with Resource Bundles
- Why Use Resource Bundles?
- The ResourceBundle Class
- How Resource Bundles Are Discovered and Named
- Property Resource Bundles
- List Resource Bundles
- Resource Bundle Caveats
- Deploying Resource Bundles with Applets
- Design Considerations for Resource Bundles

5. Formatting Messages
- Date and Time Formats
- Number Formats
- Message Formats

6. Character Sets and Unicode
- What Are Character Sets?
- What Are Encoding Methods?
- What Is Unicode?
- Unicode Encoding Methods
- Code Set Conversion

7. Searching, Sorting, and Text Boundary Detection
- Collation Issues
- Sorting in Java
- Tailoring Collation
- Improving Performance
- Searching
- Detecting Text Boundaries

8. Fonts and Text Rendering
- Characters, Glyphs, and Fonts
- Java's Font-Related Classes
- Components for Rendering Complex Text
- TrueType Font Support in Java
- Working with the font.properties File
- Adding New Fonts to Your System

9. Internationalized Graphical User Interfaces
- General Issues
- Component Orientation
- Internationalization and Localization Caveats for Various Components
- Using a Layout Manager
- Copying, Cutting, and Pasting International Text
- A Simple Example

10. Input Methods
- What Are Input Methods?
- What Is the Java Input Method Framework?
- Selecting Input Methods
- Using the Input Method Engine SPI
- Developing a Simple Input Method
- How to Package and Install Input Methods
- Developing a More Complex Input Method

11. Internationalized Web Applications
- Applets
- Servlets
- JavaServer Pages

12. Future Enhancements to the Internationalization Support in Java
- Unicode 3.0 Support
- Enhanced Complex Text
- Character Converter Framework
- Improving the Input Method Framework

A. Language and Country Codes

B. Character Encodings Supported by Java

C. Unicode Character Blocks

D. Programmer's Quick Reference

E. Internationalization Enhancements Across Versions of the JDK

Glossary

Bibliography

Index

Missed an issue?

We've got 'em all for you on CD!

JAVA DEVELOPER'S JOURNAL

The most complete library of exclusive *JDJ* articles on one CD!

Check out over 500 articles covering topics such as...
Java Fundamentals, Advanced Java, Object Orientation, Java Applets, AWT, Swing, Threads, JavaBeans, Java & Databases, Security, Client/Server, Java Servlets, Server Side, Enterprise Java, Java Native Interface, CORBA, Libraries, Embedded Java, XML, Wireless, IDEs, and much more!

JDJ The Complete Works
~~Reg $119.99~~
Buy Online
Only $71.99

XML JOURNAL

The most complete library of exclusive *XML-J* articles on one CD!

Check out over150 articles covering topics such as...
XML in Transit, XML B2B, Java & XML, The XML Files, XML & WML, Voice XML, SYS-CON Radio, XML & XSLT, XML & XSL, XML & XHTML, 2B or Not 2B, XML Industry Insider, XML Script, <e-BizML>, XML & Business, XML Demystified, XML & E-Commerce, XML Middleware, and much more!

XML-J The Complete Works
~~Reg $59.99~~
Buy Online
Only $53.99

COLDFUSION Developer's Journal

The most complete library of exclusive *CFDJ* articles!

Check out over 250 articles covering topics such as...
Custom Tags, ColdFusion and Java, Finding a Web Host, Conference Reports, Server Stability, Site Performance, SYS-CON Radio, ColdFusion Tips and Techniques, Using XML and XSLT with ColdFusion, Fusebox, Building E-Business Apps, Applicaton Frameworks, Error Handling, and more!

CFDJ The Complete Works
~~Reg $79.99~~
Buy Online
Only $71.99

SPECIAL OFFER:
Buy CFDJ & CFA The Complete Works
For Only $129.99

CF Advisor

The most complete library of exclusive *CFA* articles!

Check out over 200 articles covering topics such as...
E-Commerce, Interviews, Custom Tags, Fusebox, Editorials, Databases, News, CF & Java, CFBasics, Reviews, Scalability, Enterprise CF, CF Applications, CF Tips & Techniques, Programming Techniques, Forms, Object-Oriented CF, WDDX, Upgrading CF, Programming Tips, Wireless, Verity, Source Code, and more!

CFA The Complete Works
~~Reg $79.99~~
Buy Online
Only $71.99

JDJStore.com

Order Online and Save 10% or More!

www.JDJSTORE.com

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

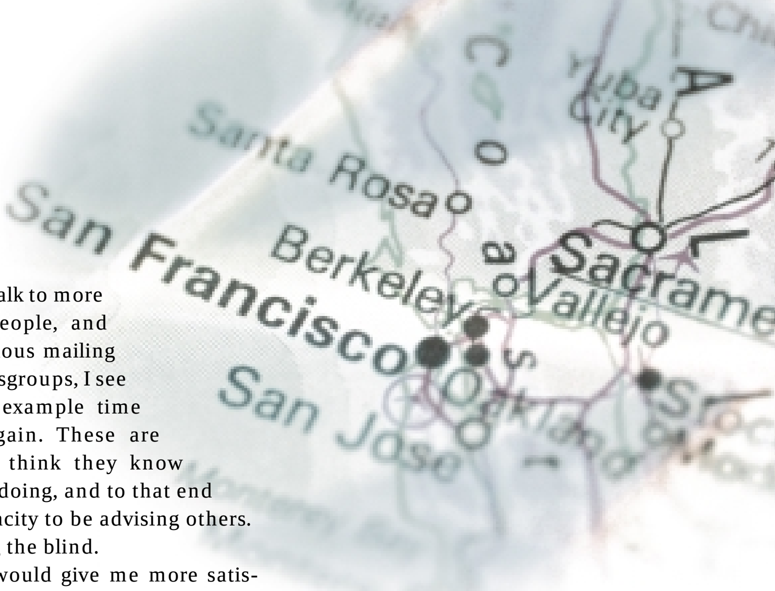
COLLECT ALL 4
FOR ONLY $229.99
JDJ, XML-J
CFDJ & CFA

web services
resource cd

www.jdjstore.com

web services resource cd

www.jdjstore.com



–continued from page 7

Java 1.4 and the Rest!

(and don't get me started on the evils of the IDE!). Using third-party tools is not a bad idea and is to be encouraged if it makes your life easier. But don't make huge leaps of faith based on them.

The post that sent shock waves through me was the one that stated that because there was only one method call being made [getRowCount()], it was much faster than all the other methods presented in this thread. Wow...major leap of the faith there, but without disparaging the poster completely, you can see the logic – well, almost! There was, of course, the plethora of posts afterwards, trying to explain to him that the third-party wrapper had no more magic at its fingertips than the JDBC driver, and therefore was probably doing the underlying [while] method of determining the number of rows. But the issue here is that this guy posted the solution in all seriousness and good faith and, at the time, really believed in what he was saying (I wonder if he was certified!).

There was, of course, a major fundamental software engineering principal lacking; as I talk to more and more people, and listen on various mailing lists and newsgroups, I see this sort of example time and time again. These are people who think they know what they're doing, and to that end have the tenacity to be advising others. Blind leading the blind.

Nothing would give me more satisfaction than having Java as the number-one programming language, but to do this we need to keep the quality up and help where we can. If you're unsure, don't post anything – you'll only lead others astray. Which is why there's always the standard question: Where can I go for good quality answers? To be honest, very few places and generally only those that are moderated. I would, however, recommend www.jguru.com, which I've found comes up trumps time and time again.

I think the moral of the tale is this: don't believe everything you read on a computer screen!

On a somewhat related note, I see Mr. Gosling headed into some controversial water last month and was quoted as saying: "IDEs are generally targeted at low-end developers – people who are not experts at writing code." It was interesting to hear him say that, particularly in this environment where the likes of ANT and Emacs are coming back into fashion. Apparently, James is working on a new kind of IDE, which I suspect he'll want to give another name to to distance his creation from the mainstream IDEs.

So with that thought, I'm off! Catch ya next month!

corda technologies

www.corda.com

SPECIAL
INTRODUCTORY OFFER
SAVE $31*
HURRY, DON'T DELAY! OFFER EXPIRES: MAY 31, 2002

Now in More than 5,000 bookstores Worldwide

FOR FAST DELIVERY

Subscribe Now!

The World's Leading Independent WebLogic Developer Resource

Helping you enable intercompany collaboration on a global scale

¥ Product Reviews
¥ Case Studies
¥ Tips, Tricks
and more!

Go Online and Subscribe Today!

WebLogicDevelopersJournal.com

SYS-CON Media, the world's leading publisher of *i*-technology magazines for developers, software architects and e-commerce professionals, brings you the most comprehensive coverage of WebLogic.
*Only $149 for 1 year (12 issues) – regular price $180.

SYS-CON MEDIA

JavaOne Japan

Sun's 2001 Worldwide Java Developer Conference

Written by Eric N. Shapiro

Zero G has participated in every JavaOne since the first one back in 1996. So, the decision to attend the first JavaOne conference held outside the U.S. was a no-brainer for us.

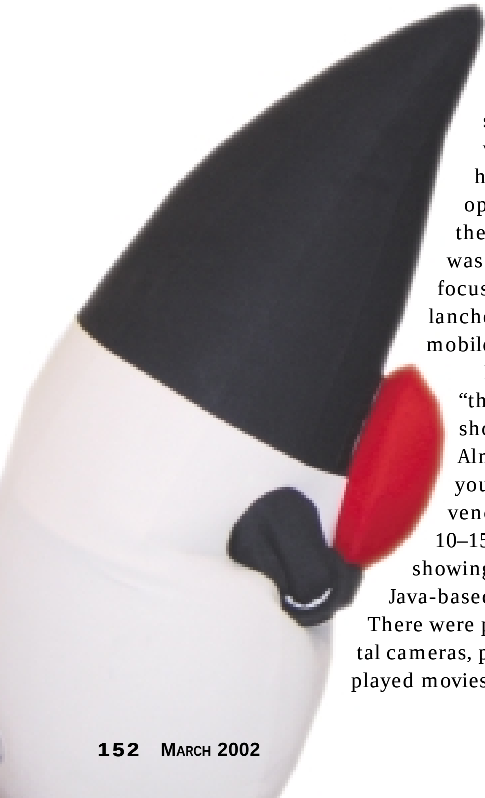
Of course, there would be some logistical hurdles to cross, but how different could it be from attending a trade show here in the states?

Boy, we'd find out.

You haven't traveled to a foreign country until you've dragged over 500 pounds of equipment with you for a trade show. Our duffle bags were crammed full of laptops, marketing collateral, posters, and demos. And drag them we did. From San Francisco to Narita Airport just outside of Tokyo, to the Narita express train, through the Yokohama train station at rush hour (not a pretty sight – six Americans dragging duffle bags upstream against tens of thousands of Japanese commuters all intent on going downstream), through the streets, and finally to our hotel.

Once we arrived at the show site, the Pacifico Yokohama conference center, we spent about three hours setting up. Looking around we saw all the usual players, including Sun, Apple, Borland, Oracle, and IBM, mostly represented by their Japanese-based offices. Most of the booths, however, were taken up by a variety of Japanese companies touting the latest for the mobile Java market.

J-Phone, NTT DoCoMo, and Fujitsu all had very large booths. Some companies were demonstrating server-side Java and app server solutions along with a good handful of developer solutions, but the real excitement was certainly focused on the avalanche of Java for your mobile phone.



If there was a "theme" to the show, it was J2ME. Almost everywhere you looked were vendor displays of 10–15 cell phones showing off a variety of Java-based technologies. There were phones with digital cameras, phones that played movies or played simulated battles or sporting contests with other Java-based phones as the owners innocently walked by each other.

At the conference section of the show, many sessions focused on designing, developing, and deploying J2ME applications. The most noteworthy sessions included Creating Mobile Services, Developing Wireless Applications with the Java 2 Platform, High Performance Java Technology for the Java Platform Micro Edition, How to Develop Impressive Mobile Applications, and Implementing the Java 2 Platform, Micro Edition Technology-Enabled Handset. Each speaker presented a well-rounded technical discussion with lots of demos and hands-on coding examples.

Java Java everywhere. Each of these phones has J2ME inside.

The session offerings were as rich and varied as those at JavaOne San Francisco, although there were a fraction as many. With over 100 keynotes, sessions, and BOFs offered, it was quite an impressive showing...not bad for a first time show (compared to over 500 at the 2001 JavaOne SF).

Sun wheeled out a full complement of bigwigs, everyone from James Gosling (father of Java) to Richard Green (VP and general manager, Java and XML software), Greg Papadopoulos (senior VP and CTO), and John Gage (chief researcher and director, science office).

And, omnipresent in the sessions, show floor, and around the conference hall was "Duke," the huge, overstuffed Java mascot, taking pictures and causing a general fuss as he lumbered from one end of the show floor to the other. Of course, the proper way to greet Duke in Japan is with a bow, and we got firsthand experience when Duke visited our booth.

Duke pauses to greet Zero G CEO Eric Shapiro and director of engineering, M. Michael Acosta

To get into the spirit of things, I rented an i-mode phone from NTT DoCoMo. Wow! There's nothing like it anywhere in the States. Just imagine a phone that's half

JAVA: INDUSTRY NEWSLETTER

JAVA DEVELOPERS JOURNAL Industry >Newsletter

Monday, December 10, 2001

BROUGHT TO YOU BY...

aligo Wireless Development Strategies - A Free White Paper. How can you use J2EE/J2ME to successfully deploy Wireless Applications? Aligo, the leader in Java based wireless application servers, offers a free technical paper on Java and Wireless Development. Click Here! for more information.

JAVA DUDES

JDJ NEWS DESK EXCLUSIVE

Oracle Targets Top Spot With New Oracle9iAS and JDeveloper

by Rick Ross - Founder of JavaLobby.org

(December 10, 2001) - I returned just yesterday from a fantastic trip to San Francisco for Oracle's OpenWorld conference. It wasn't clear what to expect when I accepted Oracle's invitation to give a presentation, but I was delighted to find the show had a high energy crowd that was very tuned in to Java, especially J2EE and Web services. The whole show was great, but the most interesting and important items came from Oracle, itself - Oracle9i Application Server Release 2 and Oracle9i JDeveloper.

WEB SERVICES: INDUSTRY NEWSLETTER

WebServices JOURNAL Industry >Newsletter

Monday, December 3, 2001

Stay Tuned for WSJ's December Issue!
Full Editorial Content Coming December 5th

BROUGHT TO YOU BY...

ORACLE Catch the Oracle9i JDeveloper wave and see how built-in profilers and Code Coach makes your Java code tighter and faster than ever before. Based on the latest J2EE and XML standards, JDeveloper combines Java coding, UML modeling, and RAD on NT/Linux/Unix. Click Here! for more information.

ONLINE EXCLUSIVE: FULL "KEYNOTE PANEL" TRANSCRIPT

Sean Rhody: Probably the question that is on almost everybody's mind is, what are Web services? I know I have a whole magazine devoted to answering that question, but I think one of the most interesting parts of the magazine is that the answers are different for different people and from different people. So to start things off I'll go down and ask each of the panelists that question. In your mind, Barry, what are Web services?

Barry Morris: I would think we all tend

XML: INDUSTRY NEWSLETTER

XML JOURNAL Industry >Newsletter

Tuesday, December 18, 2001

BROUGHT TO YOU BY...

flamenco Secure Web services. Reliable Web services. Manageable Web services. If you need Web services then you need a Web services network. Security, guaranteed delivery, and management services with no code changes. Plug into the Flamenco Network and get going! For a limited time, bring your Web services and test-drive the Flamenco Network for free.

XML NEWS

Xerces-C++ XML Parser Version 1.6.0 Provides Full Support for W3C XML Schema Recommendation

(Toronto) - The XML Parsers Development division of the IBM Toronto Laboratory has announced the much-anticipated release of the Apache Xerces-C++ XML parser version 1.6.0, including full support for the W3C XML Schema Recommendation version 1.0. Xerces-C++ is a validating XML parser written in a portable subset of C++; Xerces-C++ makes it easy to give applications the ability to read and write XML data. A shared library is provided for parsing, generating, manipulating, and validating XML documents. [illegible] features: conformance to the XML Specification 1.0, tracking of [illegible] [Level 1.0], DOM [Level 2.0], SAX/SAX2, namespace, and XML [illegible] source code, samples, and documentation; [illegible] and validation of XML; pluggable catalogs, validators [illegible] performance; and customizable error handling. The latest [illegible] C++ includes a port to FreeBSD and allows users to [illegible] through a method call. Source code and binaries [illegible] Linux, Solaris, and Windows.

[illegible] high performance, modularity, and scalability. Source [illegible] are provided with the parser. For [illegible] to make minimal use of templates, no RTTI, [illegible] minimal use of #ifdefs.

A Java-enabled gas station pump: the e-gas station

the size of anything you've ever seen, add a full-color, high-resolution screen and built-in Internet connectivity using a packet-based network – so it's always connected – and it weighs almost nothing.

While the ubiquitous "tumbling Duke" was the first application I ran (known as an i _ppli –based application), the most useful Java applications I used gave me a customized schedule of the conference sessions, and also helped us find the best local restaurants in town. But let's get back to the show itself…

One of the more interesting things at the show was the full-size gas station pump that had an embedded Java server so you could remotely control and manage the pump.

JavaOne Japan was a bit smaller than the San Francisco–based event (JavaOne SF has tens of thousands of attendees, while its Asian counterpart had just about 6,000, although this exceeded Sun's predictions). There was also, surprisingly, a complete lack of tchotchkes at JavaOne Japan. There were no blinky bouncy balls, pool ponies, or squeeze toys. And we seemed to be the only American company exhibiting that didn't have a Japanese office. Luckily, our friends at Grape City, one of our Japanese reseller partners, helped us out by providing much-needed translation and etiquette guidance.

Walking the floor, I saw how much Java technology has developed from its 1.0 beginnings. Here was a trade show floor filled with devices, from workstations and servers to mobile phones, keycards, and even gas pumps, all powered by Java technologies. Java has evolved and matured, and now powers an incredibly large percentage of the corporate and Internet economy.

The remarkable success of Java everywhere, even in consumers' pockets, is obvious without looking anywhere special. Over 4 million Java-powered mobile phones have been sold in Japan, creating an incredible opportunity for a new wave of Java solutions for everyday use. With the technology and infrastructure now largely in place, all that remains is a series of killer apps to keep us glued to our phones even more than we are already.

From my vantage point, there are only two stumbling blocks to making this next transition. First, the current Java-based devices are mostly low-powered J2ME CLDC units. These Connected Limited Device Configuration phones have very limited memory available and slow processors. Thankfully, this problem will quickly be eliminated, as evidenced by the new wave of high-powered J2ME devices that are about to hit the market. (For example, Nokia showed off a Symbian-powered device with tons of RAM and processor power.) Even the bandwidth issue has already been solved. NTT DoCoMo showed off new phones and services under the FOMA moniker that have 384 Kbps connectivity. (That's faster than my DSL connection here. In the U.S., both Sprint and AT&T have promised 3G phone connectivity by next year – we're still holding our breath.)

Second, these great new devices are not available outside of Asia. Since a considerable amount of software development occurs outside of Asia, we're stuck with a conundrum: with very few exceptions, it's nearly impossible to develop and sell software for the Japanese mobile-Java market unless you're located in Japan. For the next few years, at least, you'll have to open a Japanese office if you want to be part of this wave.

Overall, the show was well worth attending. If you find JavaOne in San Francisco useful, you should plan on attending JavaOne Japan as well.

Oh yeah, and when you get your business cards translated, make sure you have someone who really speaks Japanese review them. In our first batch my title was translated to "Most Honorable Boiler Room Mechanic."

Author Bio

Eric N. Shapiro is CEO and cofounder of Zero G Software, Inc.

eric.shapiro@zerog.com

The Zero G booth with staffers from Zero G and Grape City

Special Introductory Offer SAVE $13.89*

WebServices JOURNAL

The world's leading independent Web Services information resource

Get Up to Speed with the Fourth Wave in Software Development

Only $69.99 for 1 year (12 issues)
Newsstand price $83.88 for 1 year

- Demystifying ebXML: A Plain-English Introduction
- Authentication, Authorization, and Auditing: Securing Web Services
- Building Wireless Applications with Web Services
- How to Develop and Market Your Web Services
- Integrating XML in a Web Services Environment
- Real-World UDDI
- WSDL: Definitions and Network Endpoints
- Implementing SOAP-Compliant Apps
- Deploying EJBs in a Web Services Environment
- Product Reviews
- and much, much more!

THE NEW PROGRAMMING PARADIGM???

WebServices JOURNAL

WSJ2.COM

FULL CONFERENCE COVERAGE

Web Services EDGE conference pg. 44

SEPT. 23-26 IN NYC

A Web Services Success Story

DUPONT PROTECTIVE COATINGS

From the Editor: The Information Paradox

Product Review

Industry Insight: Mission Critical Web Services

From the Industry: What's Happening in Web Services

News pg. 64

Content Management & Web Services

eContent Services– the services behind Web Services

Page 6

Business Basics: Web Services Fundamentals: From Whence It Came A look at future possibilities Phil Karecki 18

WSJ Feature: Web Services: Enabling Technology or New Programming Paradigm? The new integration model Rebecca Dias 22

Business Collaboration: Understanding the Depth and Breadth of the Business Opportunity Revenue growth David Russell 28

Technology: Information Syndication Using Web Services A dynamic way to share data and boost profitability Darren Govoni 34

WSJ Feature: A Web Services Container A better environment for real-world deployment Ben Bernhard 38

Java: JAXM: Interoperable SOAP Communications for the Java Platform Extensibility expedites e-commerce Dave Chappell 48

Another View: The Real Niche for Web Services Part 2 Tying up the loose pieces of evolving systems Kurt Cagle 52

Open Source: Atoms vs Bits and the Digital Middle Mile Agreement on standards expedites sharing among partners Noel Clarke

UDDI: Understanding the UDDI tModels and Taxonomies The key integration point

SYS-CON MEDIA

www.wsj2.com

SYS-CON MEDIA

SYS-CON Media, the world's leading publisher of *i*-technology magazines for developers, software architects, and e-commerce professionals, brings you the most comprehensive coverage of Web services.

**Offer subject to change without notice*

OUR *i*-TECHNOLOGY FUTURE - IN JUST TWO WORDS!

EXPERT OPINIONS DIFFER: WILL IT BE A CASE OF "CHAOS REIGNS" OR OF "J2EE DELIVERS"?

by Jeremy Geelan

IN AN AGE OF SPIN AND COUNTERSPIN, *where no one calls a spade a spade if there's a chance of calling it an HDK instead ("hole development kit"),* JDJ Industry Newsletter *decided to canvass Internet technology experts of every stripe and ask them to anticipate the future – in just two words. Brevity, we felt, might increase the pressure on them to be incisive and insightful, and we weren't disappointed.*

"In two words, and only two words," we asked, "what, in your view, does the first half of 2002 hold in store for the Internet technology space?"

One well-known spokesman of a major technology company, whose identity we have politely agreed to protect, responded that "It would be easier to send 500 than two," before going on to admit that the two-word answer he'd really like to offer would probably be too candid for his (or her) masters: "Chaos reigns."

If not chaos, though, then what? Here are the answers that have been coming in from technology experts and professionals the length and breadth of the Internet.

Charles Goldfarb, the Father of XML Technology, doesn't hesitate: "XML rules!" he declares, adding that this particular prediction is "for the recently arrived Martian who might not have noticed."

James Gosling, the Father of Java, is both upbeat and awe-inspiring: "Intelligence everywhere" he intones, in what might almost be an all-purpose incantation or mantra for 2002, from one of the world's leading Internet technologists.

Tyler Jewell, director of technology evangelism for BEA Systems, is equally unequivocal, though he requires a hyphen to keep within the two-word limit. "Java-infrastructure growth" is his prediction. "The first half of 2002," he explains, "will see a renewed look at enterprise infrastructure investments by corporations...and those investments are going to made almost purely in the Java space." Fighting talk indeed.

The value-investment theme is echoed by Russell Glass, VP of strategy, AGEA: "ROI rules" is his bid. While for Simeon Simeonov, chief architect of Macromedia Inc., the two-word future we have in store is this: "Rich clients."

But Andrew Watson, VP and technical director, OMG, has a different take. "UML extends" he proclaims, referring to the Unified Modeling Language. "Use of OMG's Unified Modeling Language will expand and version 2 will be finalized," he adds.

Alan Williamson, editor-in-chief of *Java Developer's Journal*, is his usual hard-hitting self. "Nearly there" or "Coming soon" will best sum up Q1 and Q2 in 2002, he reckons.

Greg Kiessling, CEO and cofounder of Sitraka Software, is by contrast in no doubt whatsoever: "J2EE delivers" he declares. No hyphens needed there!

Both Annraí O'Toole, executive chairman of Cape Clear, and Dave Chappell, vice president and chief technology evangelist of Sonic Software, each offer the same two words. They won't surprise anyone who knows either of them: independently of one another, "Web services" is the confident prediction of each of them for what the first half of 2002 holds in store for the Internet technology space.

Yancy Lind, CEO of Lutris Technologies, comes in with a slight tweak to that. "Java services" is his two-word take. And Barry Morris, CEO of IONA Technologies, qualifies the O'Toole/Chappell prediction somewhat too, by predicting "Service architectures" rather than just Web services.

▸ Ron Worman, vice president of Global Alliances for IONA , cannily uses the two-word limit to introduce IONA's own particular brand (literally) of future, namely "E2A Integration" – we're not sure if that counts as one word, two, or maybe four, because spelled out in full it signifies "End to Anywhere" and seems to derive from IONA's corporate aphorism: "End to end is nothing. END TO ANYWHERE is everything."

▸ Another refinement is offered by Sean McGrath, CTO of XML specialists Propylon. "Dataflow desideratum" he tells us from Ireland, is what we need to watch out for. "The 'dataflow' method of designing systems was popular in the late seventies and early eighties," he explains, "and then was neglected in favor of object-oriented design. I believe XML and Web services usher in a return to focusing on distributed data flows in designing Internet-based systems."

▸ Stefan Van Overtveldt, program director of IBM's WebSphere technical marketing group, is at least honest about having been defeated by our two-word rule. "Web services breakthrough" he predicts. "Not exactly two words, but it's the best I can do," he adds cheerfully. "Companies will see the value of Web services as an open standards-based approach to the integration of multiple applications running on a variety of platforms, and start to invest in prototypes and actual deployments." He also notes that he expects application servers to become "integration servers," explaining this prediction as follows: "With J2EE 1.3 and Web services, the application server will more and more be used as a standards-based hub for integration between new and existing applications."

▸ Whereas Rick Ross, founder of JavaLobby, seems not to be at all confident in the resourcefulness of the *i*-technology sector at present. "Innovation deficit" he predicts, somewhat gloomily.

▸ Charles Arehart, founder and CTO of SysteManage, is never gloomy – "JSPs live" he offers. But being laconic is more difficult and Arehart protests to us that a three-word limit would help him since, as he puts it, "'JSPs not dead' or 'JSPs live on' connote a lot more than simply 'JSPs live' or (worse) 'JSPs rule,' which sound like so much boosterism." Charlie's prediction, in short, is that despite prognostications to the contrary, JSPs will live on well into 2002 and beyond.

▸ And regular *JDJ* columnist Blair Wyman, IBM developer extraordinaire, has a characteristically whimsical prediction, immediately understandable to anyone with a yen for entity beans. "I was puzzling about this when I sat down to supper last night – to absolutely one of my favorite meals in existence – and had a sort of epiphany," he reports. "My two-word answer is 'Bean soup.'"

The future arrives faster in the *i*-technology world than anywhere else. What's your prediction for what's ahead? Add your comments at www.sys-con.com/java/article.cfm?id=1291.

"Let the market decide"

IBM'S STEFAN VAN OVERTVELDT COMMENTS ON WEBSPHERE VS .NET DEBATE:

FOR SEVERAL MONTHS NOW, *IBM and Microsoft have engaged in an Internet debate over the superiority of their respective platforms, WebSphere 4.0 and Visual Studio .NET. You'll find details of the dialogue in the February* Java Developer's Journal *(Vol. 7, issue 2), and also at www.sys-con.com/java. Responding to an open letter by Greg Leake (group product manager, Microsoft), IBM's Stefan Van Overtveldt comments:*

"IBM and Microsoft can fight about benchmarks forever, but what matters most is what customers and developers are doing. .NET only supports Windows and other Microsoft technologies, while IBM offers tools like Eclipse and WebSphere Studio that are truly cross-platform and open standards-based. At the end of the day, it is obviously going to be hard to prove in a discussion like this whether one tool or another is more productive. The people who will make that call are the developers out there writing the code, and developers would rather have a choice of platform and vendors.

We believe that it's all about real-world scenarios, not about just one application like petstore.com. Developers and organizations will pick the environment and tools that provide the lowest total cost of ownership, while matching their particular systems requirements. Most companies don't just rely on Windows but have to work with heterogeneous systems like Unix, mainframes, etc. – which is why IBM's tools are ideal. Let the market decide."

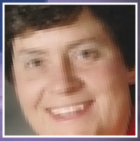
To add your comments to the discussion, go to www.sys-con.com/java/.

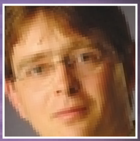
Stefan Van Overtveldt
Program Director, WebSphere Technical Marketing, IBM

Jump-start your Web Services knowledge. Get ready for Web Services Edge East and West!

AIMED AT THE JAVA DEVELOPER COMMUNITY AND DESIGNED TO EQUIP ATTENDEES WITH ALL THE TOOLS AND INFORMATION TO BEGIN IMMEDIATELY CREATING, DEPLOYING, AND USING WEB SERVICES.

EXPERT PRACTITIONERS TAKING AN APPLIED APPROACH WILL PRESENT TOPICS INCLUDING BASE TECHNOLOGIES SUCH AS SOAP, WSDL, UDDI, AND XML, AND MORE ADVANCED ISSUES SUCH AS SECURITY, EXPOSING LEGACY SYSTEMS, AND REMOTE REFERENCES.





PRESENTERS...

Anne Thomas Manes, Systinet CTO, is a widely recognized industry expert who has published extensively on Web Services and service-based computing. She is a participant on standards development efforts at JCP, W3C, and UDDI, and was recently listed among the Power 100 IT Leaders by Enterprise Systems, which praised her "uncanny ability to apply technology to create new solutions."

Zdenek Svoboda is a Lead Architect for Systinet's WASP Web Services platform and has worked for various companies designing and developing Java and XML-based products.

EXCLUSIVELY SPONSORED BY
systinet

BOSTON, MA (Boston Marriott Newton) SOLD OUT!	JANUARY 29
WASHINGTON, DC (Tysons Corner M SOLD OUT!	FEBRUARY 26
NEW YORK, NY (Doubletree Guest Suites)	MARCH 19
SAN FRANCISCO, CA (Marriott San Francisco)	APRIL 22

REGISTER WITH A COLLEAGUE AND SAVE 15% OFF THE $495 REGISTRATION FEE.

Register at www.sys-con.com or Call 201 802-3069

SYS-CON *i*-TECHNOLOGY EDUCATION

TogetherSoft™

ControlCenter is the Model-Build-Deploy Platform for building enterprise software applications. ControlCenter integrates with leading databases and application servers. It is the first end-to-end development platform to support C++, Java, and Microsoft languages. Team members can customize the user interface according to their different roles in the software development process.Together ControlCenter (TCC) reduces the learning curve because the single-development environment means there's no context switching between products. Its audits and metrics capabilities reduce errors. TCC also connects teams by eliminating the barriers for multi-development locations. TCC allows you to leverage your current resources and tools to maximize value with minimal cost.

Unify eWave

eWave Engine is an easy to embed Web Application Server for developing and deploying server-side Java technology based applications. Focused on ease-of-use, Unify eWave Engine delivers advanced performance and reliability so you focus solely on the business logic of your application. Unify eWave Engine offers seamless migration from JSP and servlets to EJB component-based development, giving customers flexibility, shortened development cycles and reduced administration burdens.

Droplets™

Droplets is a server-based application platform that provides all the administrative benefits of server-based applications, together with the functionality and usability advantages of local client software. The Droplets SDK is a distributed GUI toolkit allowing Java and C++ developers to write and deploy browser-independent desktop-style GUIs running on a lightweight and invisible Droplets client. This smart client architecture allows developers to write, test, and maintain software entirely on the server, without having to build client-server communication or port to various client platforms.

JAVA DEVELOPER'S JOURNAL

PRODUCT

SHOWCASE CD

COPRESENTED BY

TogetherSoft™

INTRINSYC

Recently identified as one of the fastest-growing technology companies in North America, Intrinsyc provides knowledge and technologies to create, link and manage pervasive networks of servers, computers and devices. Our reference designs help customers get their Internet-enabled devices to market quickly. Our bridging technologies help customers connect disparate enterprise information systems and link their embedded devices together. This includes J-Integra, which provides integration between pure Java and Microsoft COM components. As well as Ja.NET, which provides access to .NET components from Java, as if these components were implemented in Java. Our remote management software allows companies to remotely manage networks of devices and appliances.

LOOX SOFTWARE

LOOX Software, a subsidiary of Engenuity Technologies, is a leading supplier of high performance, high quality Java visualization components which increase productivity, reduce costs, and mitigate the risks associated with developing highly visual supervision and configuration applications. The JLOOX Visualization Suite is ideal for telecommunications and network management systems, workflow editors, intelligent transportation systems, and geographical information systems. Based on Java 2 Standard Edition and including support for Scalable Vector Graphics (SVG), JLOOX tools offer breakthrough performance and productivity gains for developers of graphics intensive Swing applications. Other notable features include the ability to display real-time information and seamlessly integrate with IDEs supporting Java 2.

MONGOOSE TECHNOLOGY

Creative Freedom for Portal Developers! Design: Choose components, connectors, and skins from the Web Services library or design your own. Assemble: Layout components onto a portal canvas and personalize it with skins. Drop the portal into your existing site or put web applications on your canvas. Deploy: Like a portal compiler, PortalStudio validates the portal definition. Run the portal on your laptop or staging environment. Click to deploy to production J2EE application servers. Manage: Web-based administration of user profiles, community monitoring and usage reports. Make a change? Modify or add a component then hot deploy it.

rM
realMethods

realMethods Framework is a complete implementation of the most powerful and purposeful design patterns the J2EE industry has to offer. All of the necessary design patterns required of any robust J2EE based application are already implemented by the realMethods Framework. The Framework provides support on all tiers (web, ejb, and data access). Its very purpose is not only to provide an implementation of the best J2EE design patterns, but to also encapsulate complexities and commonalities found in developing any J2EE Application.

MARCH 2002 EDITION

WWW.JAVADEVELOPERSJOURNAL.COM

SYS-CON MEDIA

Open Middleware

"Middleware for the masses"? How about "open middleware"? (see "Scandalous Propaganda" by Alan Williamson, Vol. 7, issue 1).

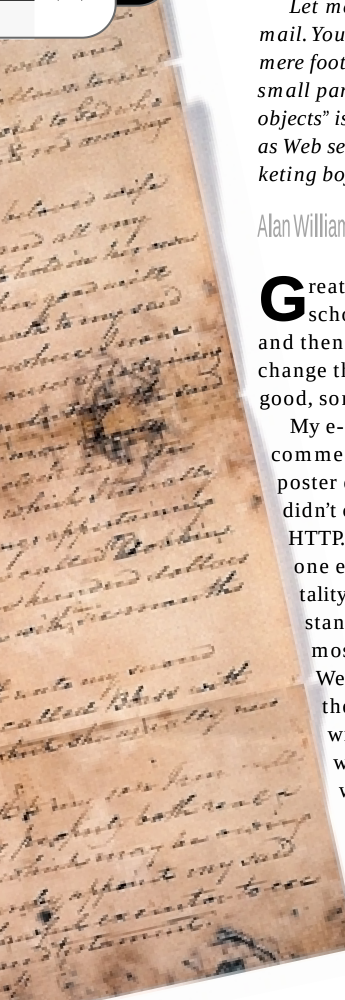
Business people are the biggest polluters of the English language, and the phrase *Web services* continues that trend by confusing the World Wide Web with the Internet. Web services has precious little to do with the WWW as far as I can tell: the WWW is an application that runs on the Internet (the "killer app" of the Internet if you will).

With the above points in mind, I would suggest "network objects", or "network object framework." BTW, I enjoy your editorials. You're able to see the forest instead of just a lot of trees.

Ron Theriault
ron-t@austin.rr.com

Let me thank you for your very kind e-mail. You're right…Web services has a mere foothold in HTTP and that's only a small part of the story. While "network objects" is a better name, it's not as catchy as Web services and to that end the marketing boys will flounder!

Alan Williamson



Great editorial. I'm from the old school – learned assembly, Cobol, and then C. I've seen a great deal of change the past 20-plus years. Some good, some not so good.

My e-mail was prompted by your comment: "At the end of day, as one poster quite correctly pointed out, we didn't consciously choose TCP/IP or HTTP. We chose them because everyone else did. It was the beehive mentality with respect to technology standards; we just swarmed to the most popular standard." I disagree. We may swarm but what makes it the most popular? I may be wrong, but I think we swarm to what "works" and because it works, it becomes popular.

I think we've chased easier ways out at times – jumping for options that promote code bloat and faster development. And Microsoft has dangled the carrot in front – showing we can be deceived and/or bought by shiny widgets. But we have to earn a living and if it means spending more time fixing something we don't have control over, eventually, we'll see the error of our ways and look for something that works.

True, development has taken on a bit of the VHS versus BETA mentality due to the Microsoft marketing machine. I admit that I have played with VB, but look at how it's changed. I totally agree with your comments about Microsoft trying to get anyone to think they can be a developer. Many people saw development as a get rich quick option. But we (the real developers) are not all alike. I know I'm not as good a developer as some of my past co-workers at WordPerfect were, but I am much better than some.

Our software world is constantly changing. I don't think a standard will win because it's popular but because it works, which makes it popular. You and I both use Java. Why? It works.

Robert Schloss
robert.schloss@fmr.com

XML Technology and Rich Browsers

I don't completely agree with everything that John Zukowski and Vincent Maciejewski said in their article "Rich User Interfaces" (Vol. 6, issue 12), apart from the fact that yes, Java is better than HTML for UIs. Personally, I think Java client-side technology development was ignored for far too long by Sun.

The one advantage of using applets over JS/HTML is the write once, run anywhere capabilities, and many of the same arguments for developing client-side GUIs still use this. That applets could run in either of the two main browsers (IE, NN) was the big advantage – you had to maintain only one piece of code.

Given that MS is determined to oust Java from its software, this advantage has dwindled for client-side Java.

When Swing came along you couldn't use it to develop applets as most JVMs installed with browsers didn't support Swing, that is, you couldn't leverage all the nice new widgets and were stuck with the boring, old AWT widget set. The AWT widget set was hardly any richer than those provided by HTML forms.

The main point is that given the current state of XML technology (XPATH, SVG, XLINK, XSD, XSLT, XSL) and its future development (XForms/UXL, etc.), I feel that Java applets are dead. Why? Because today's browsers are more standards-compliant and are becoming more capable of handling XML-based technology. For example, XSD has the model of the data from which a rich browser should be able to render the appropriate GUI. No need to transfer any GUI-rendering code to the client as the client intelligently works out the GUI for itself.

Nice article.

Sean Redmond
sean.redmond@cenorm.be

Working with Ant

Kudos to Alan Williamson for his editorial ("Back to Work with Ant," Vol. 6, issue 12) about the efficiency of lightweight tools such as Ant and text editors. To my chagrin I work with "gorilla" tools such as JBuilder and the Rational Suite everyday. I worked with Ant on the last project and will vouch for the increased productivity, especially for EJB development where you're typically utilizing an application server on your local workstation. However, when attempting to champion this approach, I'm frequently asked about debugging capabilities. I've used JDB in the past, but would admit that a graphical debugger is very nice.

Tim
tdennison@cardinalsolutions.com

SHOP ONLINE AT **JDJSTORE.COM** FOR BEST PRICES OR CALL YOUR ORDER IN AT **1-888-303-JAVA**

BUY THOUSANDS OF PRODUCTS AT GUARANTEED LOWEST PRICES!

GUARANTEED BEST PRICES
FOR ALL YOUR
WEB SERVICES
SOFTWARE NEEDS

JDJStore.com

MICROSOFT

$2,238.99 **Visual Studio .NET Enterprise Architect**

Visual Studio .NET provides developers with the most productive tool for building next-generation applications for Microsoft Windows® and the Web. Visual Studio .NET Enterprise Architect (VSEA) builds on the power of Visual Studio .NET Enterprise Developer by including additional capabilities for designing, specifying, and communicating application architecture and functionality. It enables software architects and senior developers to provide architectural guidance and share best practices across the development team.

ALTOWEB

$3,540.00 **Application Platform Release 2.5**

The AltoWeb Application Platform lets you build, deploy, and manage J2EE applications and Web services up to 10x faster without requiring extensive J2EE or Web services expertise. How? By replacing lengthy, custom and complex J2EE, XML and Web services coding with rapid component assembly and reuse.

MICROSOFT

$1,629.99 **Visual Studio .NET Enterprise Developer**

With Visual Studio .NET Enterprise Developer, developers can securely version and share their source code, share best practices, target scalable .NET Enterprise Servers, choose from a wide range of third-party tools and technologies, and easily tune the performance of their Web applications and Web Services through the extensive performance-testing tools in Visual Studio .NET.

WHIZLABS

$39.95 **WebSphere@Whiz Certification Simulator**

3 Mock Tests (159 Questions). It comes with a test engine and a question bank of 159 questions on the latest pattern of the IBM WebSphere Certification Exam. It also consists of a diagnostic test which will help you know your strengths and weaknesses so you can plan your preparation accordingly.

SITRAKA

$3,999.00 Sale Price **$3,799.00** **JClass ServerChart V1.1**

If you've used JClass Chart in your client-side applications, you already know the value of data visualization to your end-users. With JClass ServerChart, you can bring a wide range of charts and graphs right to their Web browsers, giving them real-time access to business-critical data in an intuitive environment.

SILVERSTREAM

$495.00 **Extend Application Server Developer Edition (5 User)**

SilverStream eXtend is the first comprehensive, real-world development environment for creating Web Services and J2EE applications. The seamless integration of our proven eBusiness engines and designers gives you the benefits of XML-based, enterprise-wide integration and the power to create, assemble and deploy service-oriented applications.

W W W . J D J S T O R E . C O M   OFFERS SUBJECT TO CHANGE WITHOUT NOTICE

wireless
BUSINESS&TECHNOLOGY
SYS-CON Media, the world's leading publisher of Internet technology magazines
for developers, software architects and e-commerce professionals,
becomes the first to serve the rapidly growing wireless
application development community!
Begin your unwired experience now
information for the unwired world
www.wbt2.com
SYS-CON MEDIA

Ashnasoft Releases AshnaMQ 2.0
(Fremont, CA) – Ashnasoft Corporation has announced the newest release of its highest-performing JMS server. AshnaMQ 2.0 boosts performance, provides a command-line interface, and comes with an easy-to-use monitoring console. It continues to be lightweight so it can run on PDAs and phones (J2ME).
www.ashnasoft.com

Sun Announces J2SE v 1.4
(Santa Clara, CA) – Sun Microsystems, Inc., has announced the general availability of J2SE version 1.4. The latest iteration of J2SE advances client application development with new GUI controls, accelerated Java 2DT graphics performance, expanded internationalization and localization support, new deployment options, and expanded support for Windows XP.
www.sun.com

Sitraka Announces PerformaSure 1.1
(Toronto) – Sitraka has released Sitraka PerformaSure version 1.1, a transaction-centric diagnosis tool for multitiered J2EE applications that enables e-business performance teams to measure, analyze, and maximize performance prior to application deployment.

PerformaSure's Tag-and-Follow technology reconstructs the execution path of end-user transactions, highlighting performance hot spots to accelerate the diagnosis of unacceptable e-business response times.
www.sitraka.com

NEXTEL / RIM TO CREATE BRAINIER BLACKBERRY

(Reston, VA) – Nextel Communications announced a three-way agreement with Research in Motion (RIM) and Motorola to develop a BlackBerry PDA featuring voice and data capabilities.

Operating on Nextel's national voice and packet data network, the smart phone will include Motorola's iDEN integrated digital wireless network technology enabling "always on" Internet access, text paging, and two-way radio communications.

RIM's BlackBerry wireless e-mail software will be integrated with Nextel's digital two-way radio service, text and numeric paging, and wireless Web online services. The device will also support J2ME applications.
www.nextel.com

MACROMEDIA JRUN ACHIEVES J2EE 1.3 COMPATIBILITY

(San Francisco) – Macromedia, Inc.'s, Java technology-based application server, Macromedia JRun, has passed Sun Microsystems' comprehensive J2EE 1.3 test suite.

The J2EE 1.3 compatibility program helps JRun developers deliver applications to market quickly, while leveraging Java technology's value proposition of simplified business integration, simplicity of development, and freedom of choice.
www.macromedia.com

Sybase Unveils EAServer 4.1
(Emeryville, CA) – Sybase, Inc., has released EAServer 4.1, a production-ready application server with J2EE 1.3 compatibility and next-generation support for Web services development, integration, and infrastructure.

EAServer 4.1 provides an integrated security model to enable customers to deliver secure enterprise Web services including single sign-on, role-based access control, and secure business objects out of the box.
www.sybase.com

PointBase Announces PointBase 4.2 and UniSync 4.2
(Mountain View, CA) – PointBase, Inc., has announced the latest upgrade to its family of pure Java, small-footprint database management solutions.

Enhancements to the PointBase 4.2 product suite include Autoincrement, also known as Identity Columns, and API functionality for loading and unloading SQL data to and from a database.

PointBase UniSync 4.2, the latest upgrade to the company's database and synchronization software, enables database synchronization between Java-enabled mobile devices and J2EE enterprise databases, increasing performance by almost 1,000%.
www.pointbase.com

Rational XDE Professional v2002: Java Platform Edition
(Lexington, MA) – Rational Software's Rational XDE Professional v2002: Java Platform Edition accelerates software development by eliminating the gap between design and development in Java IDEs.

It enables Java developers to code and design directly in the IBM WebSphere Studio Application Developer without switching between different, loosely integrated tools. The software also includes the IBM Eclipse IDE, providing a Java integrated development environment to developers who are not committed to a particular Java IDE.
www.rational.com

THOUGHT Inc. Ships CocoBase Enterprise O/R 4.0
(San Francisco) – THOUGHT Inc. has announced the release of version 4 of CocoBase Enterprise O/R. This new release adds support for EJB 2.0, CMP entity bean relationships, local and remote bean interfaces, extended transparent persistence support for local and distributed environments, generic session beans, and Tomcat 4.0 JSP code templates with optional user transaction objects.
www.thoughtinc.com

Nazomi Introduces Java Accelerator Chip for Wireless Apps
(Santa Clara) – Nazomi Communications Inc. has unveiled its first silicon IC product, the JA108. The new Java accelerator chip speeds up Java software execution while conserving battery life in 2G/2.5G/3G devices. It's the first in a series of product offerings under Nazomi's new KChip product line.

The JA108 works with any baseband processor, chip set, system on chip (SoC), or microprocessor, and is transparent to existing designs and legacy operating systems. It also allows designers to choose any JVM or real-time operating system.
www.nazomi.com.

✓

EnginData Presents

2002 Developer Market Survey Reports

$2,995
Java Developer Market Survey 2002
$2,495
Web Services/XML Developer Market Survey 2002
$1,995
Wireless Developer Market Survey 2002

Our comprehensive reports offer insight and strategy to guide your most critical business decisions in today's fastest growing technologies...

✓ Establish your product and marketing strategy

✓ Understand your customer's needs

✓ Evaluate technology & trends

Preview and order reports at www.engindata.com

engindata.com

engindata✓ RESEARCH

SAVE UP TO $100 ON MULTIPLE SUBSCRIPTIONS

Special online offers

Pick 4 or 5 and Subscribe for one special low price

RECEIVE YOUR DIGITAL EDITION ACCESS CODE INSTANTLY
WITH YOUR PAID SUBSCRIPTION

Wireless Business & Technology • Java Developer's Journal
Web Services Journal • WebLogic Developer's Journal
XML-Journal • WebSphere Developer's Journal
ColdFusion Developer's Journal • PowerBuilder Developer's Journal

SYS-CON MEDIA

OFFER SUBJECT TO CHANGE WITHOUT NOTICE

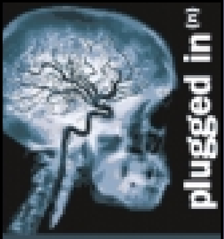
WWW.SYS-CON.COM/SUBOFFER.CFM

J2ME
J2SE
J2EE
Home
Online

DOT.COM
- buyer's guide
- java forums
- mailing list
- java jobs
- java store

MAGAZINE
- advertise
- authors
- customer service
- editorial board
- subscribe

CONTENT
- archives
- digital edition
- editorial
- features
- interviews
- product reviews
- source code



THE JAVA DEVELOPER CONNECTION℠ PROGRAM
Get pre-release Java™ technology downloads
REGISTER TO GET THEM
Java Sun

What's Online...

March **2002**

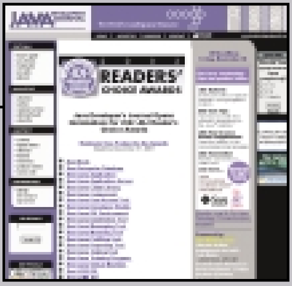
***JDJ* Online**

Visit www.javadevelopersjournal.com every day for fast-breaking Java news and events. Know what's happening in the industry, minute by minute, and stay ahead of the competition.

2002 Readers' Choice Awards

Vote for your favorite Java software, books, and services in our annual ***JDJ*** Readers' Choice Awards. Categories include Best Java Testing Tool, Best Mobile Database, Best Java Messaging Tool, Best Java E-Business Framework, Best Java EAI Platform, Best Java Web Services Development Toolkit, and Best Database Tool/Driver.

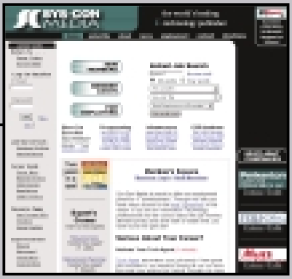
The ***JDJ*** Readers' Choice Awards program, often referred to as the "Oscars of the software industry," has become the most respected industry competition of its kind. Winners will be announced in June at Web Services Edge 2002 East Conference and Expo in New York City.



Web Services Edge 2002 East
Coming to NYC, June 24–27

SYS-CON Events' Web Services Edge 2002 East International Web Services, Java, and XML Conference & Expo will take place June 24–27, 2002, at the Jacob Javits Convention Center in New York City. The event will complement the Technology Exchange Week New York (TECHXNY), a weeklong showcase for the most dynamic business technology players and products in the world.

For further information go to www.sys-con.com/WebServicesEdge2002East

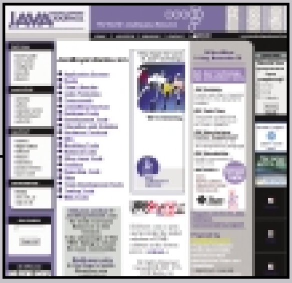


Search Java Jobs

Java Developer's Journal is proud to offer an employment portal for IT professionals. Get direct access to the best companies in the nation. Learn about the "hidden job market" and how you can find it. If you're an IT professional curious about the job market, this is the site to visit.

Simply type in the keyword, job title, and location and get instant results. You can search by salary, company, or industry.

Need more help? Our experts can assist you with retirement planning, putting together a résumé, immigration issues, and more.



JavaBuyersGuide.com

JavaBuyersGuide.com is your best source anytime, anywhere on the Web for Java-related software and products in more than 20 mission-critical categories, including application servers, books, code, IDEs, modeling tools, and profilers. Check the Buyer's Guide for the latest and best Java products available today.

WAKESOFT Learn more now!

Click for a FREE 30-day trial AltoWeb

M-1 Mobile App Server aligo Download Now!

sitraka

Get SonicMQ® sonic SOFTWARE

MERANT DataDirect

The Truth About Web Services IONA Webcast Series

KADA

spiritsoft going beyond jms

InstallAnywhere PowerUpdate ZERO G www.ZEROG.com Click here to try

sys-con events editorial

www.sys-con.com

sys-con events editorial

www.sys-con.com

sys-con events editorial

www.sys-con.com

sys-con events editorial

www.sys-con.com

sys-con events editorial

www.sys-con.com

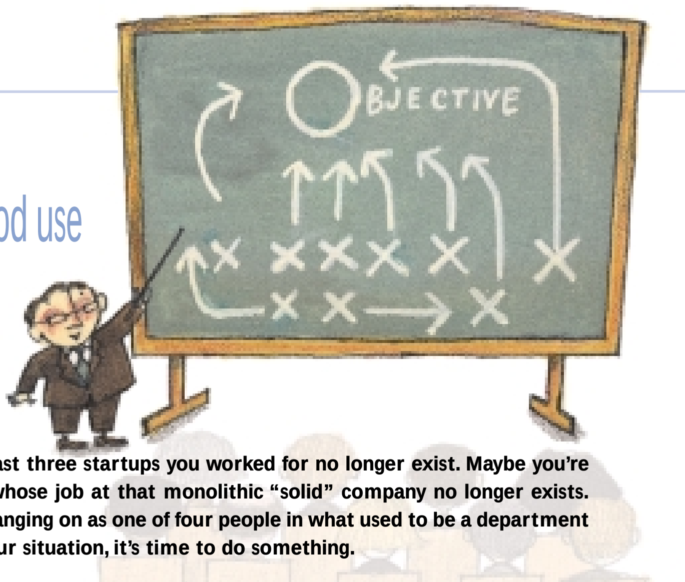
sys-con events editorial

www.sys-con.com

Back to School

Put downtime to good use

WRITTEN BY
BILL BALOGLU & BILLY PALMIERI



Maybe the last three startups you worked for no longer exist. Maybe you're one of the thousands whose job at that monolithic "solid" company no longer exists. Or maybe you're still hanging on as one of four people in what used to be a department of 30. Regardless of your situation, it's time to do something.

As the high-tech industry and the national economy attempt to inch toward recovery, many tech professionals are taking a good hard look at their skills, defining new goals for their future, and using downtime to return to school.

For junior or intermediate Java programmers, the path of least resistance may be to get Java certified through an online training program. These computer-based training programs can certainly help you expand and refine your skills, but Java certification doesn't necessarily guarantee that new doors of opportunity will suddenly fly open for you.

Any type of technical training adds to your professional cachet, but we have yet to see a hiring manager specifically request that candidates be Java certified or break into a dance of joy to find it on a candidate's résumé.

What hiring managers *do* respond to are bachelor's and master's degrees in computer science and/or electrical engineering from colleges known for their technical programs.

Going back to school for two to four years may not be feasible for everyone, but unlike employment, education is something that can never be taken away from you.

If you have a solid Java background and are looking to make it stronger, you might want to seek out courses in skills that are currently in demand, such as knowledge of distributed systems, enterprise applications, multithreading, scalability, and security.

For a broader perspective on these questions of technical training, we consulted someone who's moved throughout technology's academic and professional worlds.

After earning a bachelor's degree in computer science at Harvey Mudd College, Craig Persiko worked at a variety of companies as a programmer, returned to school to get his master's degree in computer science at NYU, and now teaches programming at City College of San Francisco.

His early exposure to the professional world was an eye-opener. "A lot of professional programmers don't have formal training, which means there's a lot of bad code out there," he said. "I was surprised how many people in the professional world picked up their programming skills on the job."

Although academia can be somewhat removed from the day-to-day realities of the professional world, Persiko attributes key skills to a formal computer science education.

"The formal study of design skills – planning things before you start coding – and issues of readability and maintainability are very important," he said. "It's important not just that the code works, but that it's easy to understand and maintain.

One area in which the academic world has had to catch up with the professional world is in the use of Java itself. Persiko's undergraduate training focused on C++.

"As a graduate student at NYU, I was amazed how many students were learning Java to start with, but it really makes sense," he said. "Java is easier than C++ to learn first. It's written to be more robust and doesn't let you make the mistakes you can in C++."

Persiko sees an increasing acceptance of Java as an academic language. There are still more students learning C++, but the programming courses he teaches are now offered in both C++ and Java.

The student population moving through the academic world has changed as well. While most of his day students are typical college-age students, people attending the evening courses tend to be working adults who've worked in some programming-related capacity but are looking for career enhancement.

A lot of them have been recently laid off and some are preparing for an IS management degree. But most of them are not coming from a traditional computer science background.

Enrollment is down for some of the Web design and graphics courses, but is strong and growing for the programming and Unix courses. "The perception that computer science is the hot career to go into has changed," he said. "It's a huge transition from a year and a half ago."

His advice to technical professionals at this juncture? "Examine what you want more of and what you want to improve on. Set your goals and decide exactly what it is that you want to accomplish."

Noting that not everyone has the time or motivation to embark on a full degree program, Persiko stresses that it's important to stay focused and take courses that will move you closer to your goals.

"Either way," he said, "It's a very good time to go back to school."

jdjcolumn@objectfocus.com

AUTHOR BIOS
Bill Baloglu is a principal at ObjectFocus (www.ObjectFocus.com), a Java staffing firm in Silicon Valley. Bill has extensive OO experience and has held software development and senior technical management positions at several Silicon Valley firms.

Billy Palmieri is a seasoned staffing industry executive and a principal at ObjectFocus. His prior position was at Renaissance Worldwide, where he held several senior management positions in the firm's Silicon Valley operations.

SUMMIT Information Systems

Corvallis, OR

www.summitsite.com

At Summit, we make money move. From deposits at the ATM to the lightning fast movement of e-commerce, we make it all work– fast, efficiently, accurately.

Java Developers & Engineers

Applicants must have 3+ years of Java and/or VB development experience. Experience with XML, XSL, SOAP, JSP, HTML/DHTML, J2EE is required. Oracle experience is a plus.

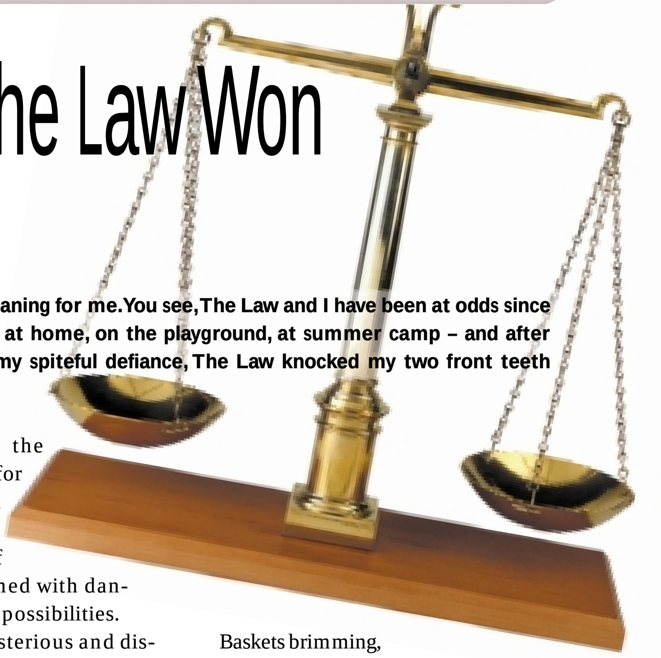
Here, in our "Silicon Forest", you'll discover a challenging, technology-rich environment surrounded by beautiful mountains and abundant hiking, biking, and recreational opportunities. **Relocation assistance may be available.**

For immediate consideration, please email your resume to:
resume@summitsite.com or fax: 541-758-4073. EOE

JAVA DEVELOPER'S JOURNAL ADVERTISERINDEX

ADVERTISER	URL	PHONE	PAGE
Actuate Corporation	www.actuate.com/info/jbjad.asp	800-884-8665	79,117
Altaworks Corporation	www.altaworks.com	603-598-2582	127
Altova	www.altova.com		39
AltoWeb	www.altoweb.com		37
AppDev Training Company	www.appdev.com/promo/MG00052		70
Apptricity	www.apptricity.com	214-596-0601	67
Aquarius Solutions	www.aquariussolutions.com		102
Ashnasoft Corporation	www.ashnasoft.com		101
BEA	developer.bea.com		4,5
Blaze Advisor From HNC	www.blazesoft.com		35
Borland Software Corp.	www.borland.com/new/optimizeit/94000.html		105
Canoo Engineering AG	www.canoo.com/ulc/	41 61 228 94 44	27
Capella University	www.capellauniversity.edu	1-888-CAPELLA	65
Compoze Software	www.compoze.com/jdj	866-266-7693	97
Compuware Corp.	www.compuware.com/products/optimalj	800-468-6342	123
Corda Technologies	www.corda.com		150
DataDirect Technologies	www.datadirect-technologies.com	800-876-3101	23
Dice.com	www.dice.com		61
EnginData Research	www.engindata.com		163
ESRI	www.esri.com/arcims	888-289-5084	29
eXcelon Corporation	www.exln.com	800-962-9620	13
Fiorano Software	www.fiorano.com	800-663-3621	107
Hewlett-Packard Company	www.hpmiddleware.com/download	856-638-6000	17
HiT Software	www.hitsw.com/xmlrdb	408-345-4001	89
IBM	ibm.com/websphere/ibmtools		30
IBM	ibm.com/websphere/ibmtools		31
IBM	ibm.com/db2/outperform		33
ILOG	www.ilog.com/jdj/jrules	1-877-223-ILOG	69
ILOG	www.ilog.com/jdj/jviews	1-877-223-ILOG	129
InetSoft Technology Corp.	www.inetsoft.com	888-216-2353	109
Infragistics, Inc.	www.infragistics.com	800-231-8588	14-15,121
Insession Technology	www.insession.com	800-755-1596	55
InstallShield Software Corp.	www.installshield.com	847-240-9111	71
INT, Inc	www.int.com	713-975-7434	46
Intraware	www.intraware.com/portal	800-469-4313	81
iSavvix Corporation	java.isavvix.com		77
Java Developer's Journal	www.sys-con.com	800-513-7111	136,143
Java Developer's Journal Product Showcase CD	www.javadevelopersjournal.com		158-159
JDJEdge Conference & Expo	www.sys-con.com	201-802-3057	137
JDJ Store	www.jdjstore.com	201-802-3012	147
John Wiley & Sons	wiley.com		59
LOOX Software Inc.	www.loox.com	800-684-LOOX	75, 131
Lutris Technologies, Inc.	www.lutris.com/javaone	877-688-3724	45
Metrowerks Corp.	www.wireless-studio.com		8
Mongoose Technology	www.portalstudio.com		51,119
New Atlanta Communications	www.newatlanta.com		87
Northwoods Software Corporation	www.nwoods.com	800-226-4662	116
Oracle Corporation	www.oracle.com/hurwitz	800-633-1072	63
ParaSoft Corporation	www.parasoft.com/jdj3	888-305-0041	49
PointBase, Inc.	www.jdj5.pointbase.com	877-238-8798	91
Pramati Technologies	www.pramati.com	877-PRAMATI	113
Precise Software	www.precise.com/jdj	800-310-4777	21
preEmptive Solutions	www.preemptive.com	800-996-4556	85
Prentice Hall PTR	www.phptr.com		53
QUALCOMM Incorporated	www.brew2002.com		99
Quintessence Systems Limited	www.in2j.com		83
Rational Software	www.rational.com/offer/javacd2		47
ReportMill Software, Inc.	www.reportmill.com	214-513-1636	133
Sharp	http://developer.sharpsec.com		115
SilverStream Software	www.silverstream.com/challenge	888-823-9700	41
Simplex Knowledge Company	skc.com	845-620-3700	135
Sitraka Software	www.sitraka.com/jclass/jdj	888-361-3264	25
Sitraka Software	www.sitraka.com/jclassSS/jdj	888-361-3264	73
Sitraka Software	www.sitraka.com/performance/jdj	888-361-3264	176
Softwired, Inc.	www.softwired-inc.com	41-14452370	103
Sonic Software	www.sonicsoftware.com	800-989-3773	2
SpiritSoft	www.spiritsoft.net/climber		43,125
SOLUTIONSsite	www.solutionssite.com	877-838-9500 x122	116
Sun Microsystems	www.sun.com/forte		11
Sun Solutions	www.solutionssite.com		111
Sybase	www.sybase.com/easerver4dev.		19
SYS-CON Custom Publishing	www.sys-con.com	925-244-9109	165
SYS-CON Industry Newsletters	www.sys-con.com	201-802-3020	153
SYS-CON Reprints	www.sys-con.com	201-802-3026	173
SYS-CON Subscription Offer	www.sys-con.com/suboffer.cfm		163
Thought Inc.	www.thoughtinc.com	415-836-9199	95
TogetherSoft Corporation	www.togethersoft.com/1/jd.jsp	919-833-5550	6
Visual Mining	www.visualmining.com/jdj	800-308-0731	93
WebGain, Inc.	www.webgain.com/toplink_create3.html	1-877-WebGain Ext.15858	175
WebLogic Developer's Journal	www.sys-con.com/weblogic	800-513-7111	151
Web Services Edge Conference & Expo	www.sys-con.com	201-802-3057	141
Web Services Edge World Tour 2002	www.sys-con.com	201-802-3069	156-157
Web Services Journal	www.sys-con.com	800-513-7111	144,155
Web Services Resource CD	www.jdjstore.com	201-802-3012	148-149
WebSphere Developer's Journal	webspheredevelopersjournal.com	800-513-7111	145
Wireless Business & Technology	www.sys-con.com	800-513-7111	142,161
XMLEdge Conference & Expo	www.sys-con.com	201-802-3056	139
XML-Journal Subscribe	www.sys-con.com	800-513-7111	138
Zero G	www.zerog.com	415-512-7771	3

I Fought The Law and The Law Won



WRITTEN BY
BLAIR WYMAN

This great Sonny Curtis tune has special meaning for me. You see, The Law and I have been at odds since I was very young. I fought The Law constantly – at home, on the playground, at summer camp – and after years of openly and repeatedly demonstrating my spiteful defiance, The Law knocked my two front teeth squarely down my throat.

I was only eight-years old but I had it coming, and I must admit, The Law was applied justly, swiftly, and with absolute fairness: that Law we know as *Gravity*.

It couldn't have been a more beautiful autumn day as the promise of a warm breeze rustled newly fallen September leaves. My pals and I jumped on our two-wheelers and rode with that breeze into an afternoon adventure.

There was nothing to do, as usual – at least, nothing that had to be done before supper. It was a weekday and none of the three TV stations showed cartoons until 5:00 p.m. The school building was closed up tighter than a drum so the teachers could go to their statewide meeting.

Weekdays out of school were always an event, but being outside that lovely day was special, like exacting a gentle sort of revenge. While my erstwhile educational "captors" sat indoors somewhere – probably fidgeting at the words of some pedagogical pundit and wishing nothing more than to trade places with me – I stood at the prow of my two-wheeled Schwinn steamer, daring the world to cross my windswept wake.

There were any number of pursuits my friends and I could have undertaken on that perfect, perfect day. There was frog hunting with slingshots by the Scary Yellow House, or killing pop cans with The BB Guns of Extreme Prejudice, or even strolling to the neighborhood grocery for frozen pops (5¢) and a glowering scowl from the proprietor (free of charge). The day brimmed with dangerous and destructive possibilities.

One of the most mysterious and distant places we ever dared visit was a place called *The Cut*. The Cut was simply a wide gash through a shale ridge that ran near the local college. The result of the original excavation was a few acres of scrubby, flattened shale at the center of a deep gash. However, there was treasure here, too. That's what made The Cut special.

On one dark little spot, deep in The Cut, oxidizing ill-fated Detroit-iron carcasses lay delinquently begging for juvenile dismemberment. Being treasure hunters one and all, we descended on the unsheeted metal frames like cackling, chrome-chomping vultures.

My Schwinn Typhoon was a real workhorse, equipped with large wire baskets on either side of the rear wheel and one on the handlebars. These baskets could hold a veritable fortune in worthless junk, as long as it was stacked carefully, and stack it we did. I don't remember exactly what was in those baskets that fateful day, but we couldn't wait to get home to tear each piece apart. Oh, the simple pleasures of destructive discovery.

Baskets brimming,
I raced the sun across The Cut.
My pals and I had tarried a bit too long, and the sun's angle promised serious repercussions if we didn't hurry home. Just as my tires came off the crazed, hardpacked shale of The Cut and onto a gravel-slickened concrete roadway, my trusty Schwinn lost its metal mind.

That traitorous twenty-four-inch two-wheeler wobbled, wiggled, and wantonly pitched me headlong over the handlebars and onto the ground. I tried to catch myself with my hands, but The Law's ardor for inertia ensured that my first kiss was shared with oil-soaked concrete. My two front teeth were never seen again.

It was a hard but fair lesson that The Law taught me that fateful autumn day. Blinded by the promise of beautiful junk, I tried to carry more than my little frame could handle. Of course, I'm bigger now and I happen to know of a great little junkyard over by Byron. Maybe if I strapped that bumper to my fanny-pack...

blair@blairwyman.com

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

Java Dudes

www.vladkolarov.com
WAM!!!!
MONTHLY EDITORIAL MEETING OF JDJ AND .NET PROGRAMMERS JOURNAL

webgain, inc.

www.webgain.com/toplink_create3.html